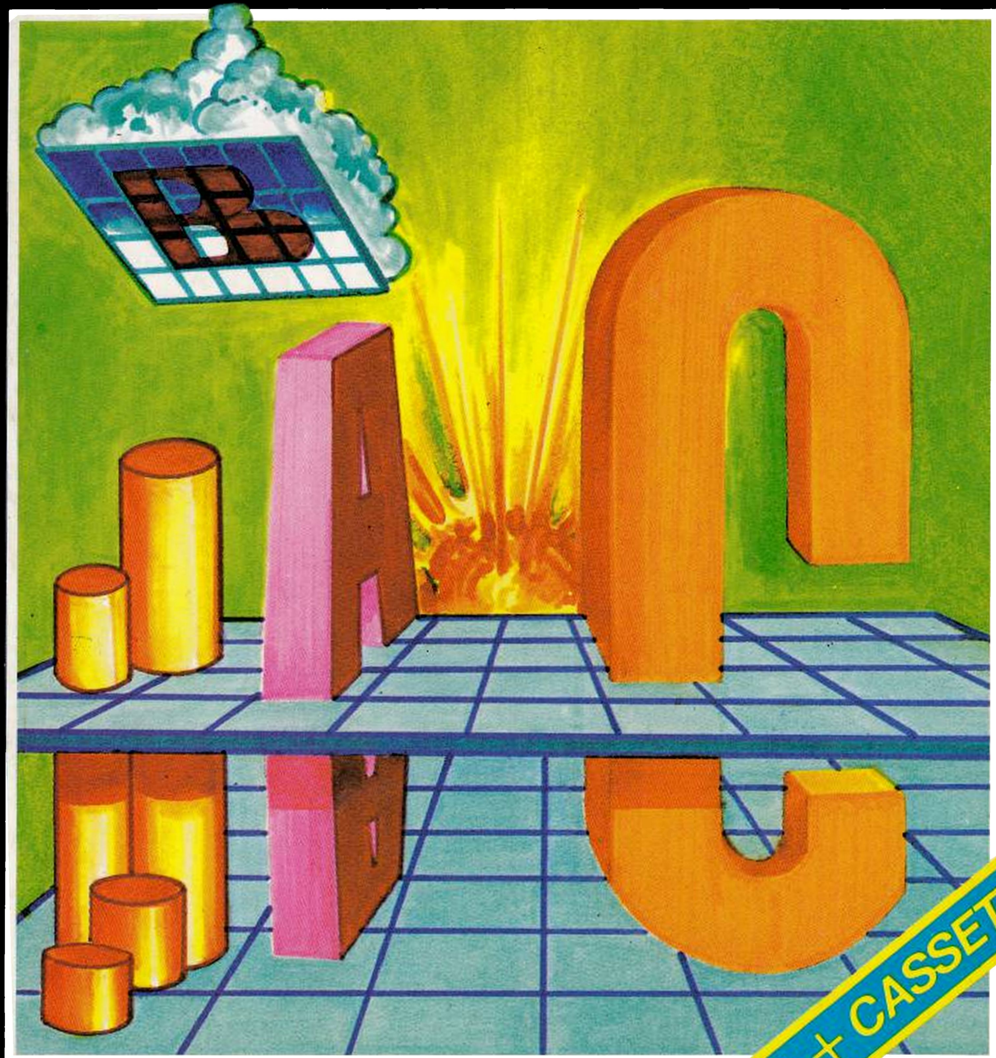
L'ABC DEL LINGUAGGIO MACCHINA PER IL C 16

La mappa completa della memoria dettagliata e commentata

di PETER BERESFORD



LIBRO + CASSETTA

edizioni JCE

L'ABC DEL LINGUAGGIO MACCHINA PER IL C 16

La mappa completa della memoria dettagliata e commentata

di PETER BERESFORD

Traduzione di F. FRANCIA



JACOPO CASTELFRANCHI EDITORE
Via dei Lavoratori, 124
CINISELLO BALSAMO (MI)

Pubblicato per la prima volta in Gran Bretagna dalla:

Melbourne House (Publishers)Ltd 1985

Titolo originale: C16 Machine Language for the Absolute Beginner

Prima edizione: Ottobre 1985

I programmi inseriti in questo libro e nella cassetta allegata hanno scopo esemplificativo ed educativo. Sono stati verificati attentamente ma non sono garantiti per alcuno scopo particolare. Nonostante sia stata presa ogni precauzione l'editore non può essere ritenuto responsabile per errori che potrebbero avvenire nell'esecuzione.

Fotocomposto elettronicamente da: JCE S.A.S.

Stampato in Italia da:
Gemm Grafica S.r.l.
Via Magretti
Paderno Dugnano (MI)

SOMMARIO

Prefazione

E così, avete da poco acquistato il vostro Commodore 16, ed utilizzate il linguaggio BASIC residente per scrivere semplici programmi. Lentamente, giorno dopo giorno, vi addentrate sempre più nell'esplorazione e nella sperimentazione del vostro nuovo computer.
Probabilmente avrete già avuto occasione di far eseguire alcuni programmi commerciali, come word processors, sistemi di contabilità, software educativo oppure videogiochi. Vi sarete senz'altro domandati cosa rende così differenti tali programmi rispetto ai vostri scritti in BASIC; essi infatti danno l'impressione di gestire più funzioni simultaneamente, incluse quelle che non avreste mai immaginato il vostro computer potesse effettuare.
Senza considerare l'ampiezza dei programmi, nè tantomeno il tempo impiegato per la loro creazione, la principale differenza fra quelli scritti da voi e quelli che potete acquistare in un negozio specializzato è rappresentata dal fatto che questi ultimi sono per la maggior parte composti parzialmente od interamente in linguaggio macchina. L'impiego del linguaggio macchina è di rigore per ogni programmatore professionista. La stragrande maggioranza dei videogiochi, molte utilità e numerosi programmi d'interfacciamento sono infatti scritti in tale linguaggio.
Lo scopo di questo libro è quello d'introdurre anche voi nell'affascinante mondo del linguaggio macchina, l'altra faccia del Commodore 16.
Sarete inizialmente condotti con estrema prudenza alla scoperta delle istruzioni del microprocessore 7501, che verranno in seguito approfondite ed illustrate per mezzo di numerosi esempi. Farete progressivamente conoscenza di molte nuove caratteristiche del vostro computer, alcune delle quali vi stupiranno decisamente.
Verrete incoraggiati attraverso la lettura a verificare che quanto il computer vi comunica in uscita corrisponda esattamente alle vostre logiche aspettative. Mantenete sempre a portata di mano carta e penna, in modo da prendere appunti strada facendo.
Al termine del libro troverete alcune appendici esaurientemente commentate e dettagliate, alle quali spesso e volentieri farà riferimento il testo; inoltre, è stato previsto un piccolo glossario comprendente i termini più frequentemente utilizzati.

CAPITOLO 1

Introduzione al linguaggio macchina

Uno dei vantaggi del linguaggio macchina è quello di permettere al programmatore l'effettuazione di determinate funzioni per le quali il BASIC non è indicato. Il suo principale pregio è tuttavia costituito dalla notevole velocità; nel Commodore 16 è infatti possibile eseguire approssimativamente centomila istruzioni al secondo. I comandi BASIC sono svariate centinaia di volte più lenti.
Ciò è dovuto al fatto che il BASIC è scritto a sua volta in linguaggio macchina, è pertanto ogni suo comando può essere considerato un vero e proprio programma composto anche da alcune centinaia d'istruzioni. Questa caratteristica si riflette in tutti i linguaggi strutturati.
Le istruzioni del linguaggio macchina, come potrete rendervi conto personalmente, sono estremamente limitate per quanto riguarda la funzione assolta. Esse svolgono appunto compiti elementari, per cui è necessario combinarne molteplici al fine di ottenere un risultato tangibile. Le loro funzioni si riferiscono direttamente all'effettiva architettura interna del computer; esse provvedono fra le altre cose a segnalare al calcolatore quali numeri ricordare e quali dimenticare, determinare se un tasto è stato premuto, leggere e scrivere informazione su disco o nastro, stampare caratteri sullo schermo.
I programmi in linguaggio macchina possono essere inoltre considerati come delle subroutines, esattamente come in BASIC, ovvero un programma nel programma richiamabile in qualunque momento, ed al termine del quale il controllo viene restituito al comando immediatamente successivo alla chiamata.

Utilizzo di un programma in linguaggio macchina

Per richiamare una subroutine in linguaggio macchina da un programma BASIC si utilizzerà il comando [SYS indirizzo]. Analogamente a GOSUB, è necessario segnalare al computer il punto di partenza della subroutine; così come GOSUB 1000 richiama la subroutine BASIC avente inizio alla linea di programma 1000, SYS 1000 richiamerà la subroutine in linguaggio macchina situata a partire dall'indirizzo di memoria decimale 1000.
Notate che un indirizzo di memoria è ben differente da un numero di linea; esso rappresenta infatti l'effettivo "indirizzo" di una specificata zona di memoria del computer, ovvero la sua locazione iniziale.

Indirizzamento della memoria

Saprete senz'altro che il C16 possiede 16 Kbytes di memoria. 16 Kbytes rappresentano il numero totale delle singole locazioni contenute nel computer. Ognuna può essere idealmente considerata come una scatoletta contenente un solo carattere, una sola informazione.
Disponendo di oltre sedicimila scatolette separate, il computer deve disporre necessariamente di un sistema di localizzazione, in modo tale da poter ritrovare agevolmente separate informazioni in qualunque momento. Questo sistema consiste nell'assegnare a ciascuna scatoletta un personale indirizzo, concettualmente simile a quello di casa vostra; tale indirizzo viene infatti utilizzato per il ritrovamento di una particolare casa in una città affollata, per l'invio di corrispondenza, oppure per il prelevamento di un pacchetto a voi destinato. Il computer, come noi, invia informazioni e si sposta da un luogo (subroutine) ad un altro, utilizzando il suo proprio sistema d'indirizzamento. Questo sistema è senza dubbio più semplice del nostro, in quanto si limita a numerare progressivamente da 0 a 65535 ogni singola locazione di memoria; per noi tutto ciò non risulterebbe molto semplice da ricordare, ma per il computer rappresenta esattamente il suo logico modo di agire. Queste scatolette numerate possono essere assimilate a delle caselle postali; nel caso abbiate depositato qualcosa nella casella numero uno, questo vi resterà fino ad un vostro successivo intervento. In ogni scatoletta potrà essere posto un unico oggetto alla volta: inserendone uno nuovo, si perderà automaticamente e per sempre il precedente contenuto.
Il comando SYS 1000 dice al BASIC di andare ad eseguire una subroutine in linguaggio macchina la cui prima istruzione è immagazzinata nella locazione posta all'indirizzo decimale di memoria 1000.

Gestione della memoria direttamente dal BASIC

Esistono altri due comandi BASIC che troverete estremamente utili nella gestione della memoria. Essi ci permettono d'immagazzinare e prelevare i rispettivi valori contenuti in ciascuna locazione. Questi comandi sono PEEK e POKE.

```
PRINT PEEK(5000)
```

provvederà a leggere il contenuto del byte situato all'indirizzo 5000 ed a stamparlo quindi sullo schermo. Tale comando può essere utilizzato come qualsiasi altra funzione all'interno di un programma BASIC; ad esempio:

```
A=PEEK(387)
```

assegna alla variabile numerica A il valore della locazione d'indirizzo 387.

```
POKE 1100,27
```

equivale idealmente all'opposto del precedente comando, ovvero memorizza il valore a destra della virgola (in questo caso 27) nella locazione il cui indirizzo è specificato a sinistra di quest'ultima (per cui 1100).
Per rendervi conto del loro funzionamento, provate a digitare quanto segue:

```
PRINT PEEK(5000)
POKE 5000,200
PRINT PEEK(5000)
```

Utilizzeremo enormemente questi due comandi BASIC nel corso dei capitoli successivi. Il BASIC costituirà un importante attrezzo con il quale scrivere, eseguire ed osservare i nostri programmi in linguaggio macchina.

Linguaggio macchina come subroutine

Abbiamo precedentemente visto come un programma in linguaggio macchina possa essere impiegato esattamente come una subroutine in BASIC. Al posto del comando GOSUB, faremo uso di SYS. Come ben saprete, una subroutine in BASIC deve terminare assolutamente con il comando RETURN. Nello stesso modo, le nostre routines in codice macchina devono concludersi con un'apposita istruzione che restituisca il controllo al programma BASIC principale. L'equivalente di RETURN in linguaggio macchina è 96. Questo valore numerico, 96 appunto, è ciò che il microprocessore interpreta come una richiesta di ritorno da una subroutine. Sarebbe praticamente impossibile per noi ricordarsi la funzione svolta da 96, così come quella di centinaia di altre istruzioni, per cui è stato deciso di dare un nome vero e proprio ad ognuna di esse. Questi nomi risultano incomprensibili al computer, ma fortunatamente hanno un senso per noi, i suoi programmatori; essi sono brevi, semplici, espressivi, ed identificati con il termine di "codici mnemonici".
Il codice mnemonico di 96 è RTS, contrazione di ReTurn from Subroutine (ritorno da una subroutine). Quando necessario, provvederemo a presentarvi tanto il valore in codice macchina quanto il suo mnemonico associato, in modo tale da rendervi comprensibile il significato delle istruzioni fornendo simultaneamente al computer l'informazione che gli compete.
Per illustrarvi il funzionamento di tutto ciò, costruiremo insieme un brevissimo programma in linguaggio macchina. Digitate la seguente linea BASIC:

```
POKE 8192,96
```

Con questa linea memorizzeremo il valore 96, corrispondente ad un ritorno da una subroutine (RTS), nella locazione di memoria 8192.
Congratulazioni! Avete appena terminato la creazione del vostro primo programma in linguaggio macchina. Non si può certo dire che faccia granchè; equivale sostanzialmente in BASIC alla chiamata di una subroutine vuota:

```
GOSUB 400
400 RETURN
```

Provvederemo adesso a far eseguire il nostro programma per mezzo del comando SYS, in modo tale da verificarne il funzionamento. Digitate la seguente linea BASIC:

```
SYS 8192
```

Il computer dovrebbe rispondervi con il messaggio READY; avrà allora appena ultimato l'esecuzione del programma.

Riassunto del Capitolo 1

[1] Il linguaggio macchina è estremamente veloce. Permette inoltre l'accesso a funzioni interne del computer non agevolmente utilizzabili tramite il BASIC.

[2] Ogni sua istruzione si limita a svolgere un compito elementare molto ridotto.

[3] La memoria viene indirizzata utilizzando i numeri da 0 a 65535 compresi.

[4] Un indirizzo di memoria può essere assimilato ad una casella postale, la quale può contenere soltanto un'informazione alla volta.

[5] Il comando PEEK viene utilizzato per esaminare il contenuto di una particolare locazione di memoria.

[6] Il comando POKE viene utilizzato per assegnare un determinato valore numerico ad una particolare locazione di memoria.

[7] Il comando SYS viene utilizzato per far eseguire un programma in linguaggio macchina dal BASIC.

[8] Il valore 96 (RTS) deve essere posto al termine di ogni programma in linguaggio macchina, in modo da segnalare al computer di effettuare un ritorno da una subroutine.

CAPITOLO 2

Le basi della programmazione in linguaggio macchina

Gestione della memoria dal linguaggio macchina

Abbiamo visto in precedenza come sia possibile intervenire sul contenuto della memoria a partire dal BASIC, leggendo e modificando i valori delle sue locazioni. Questo è ovviamente fattibile anche all'interno dei nostri programmi in linguaggio macchina. Dovremo essere in grado di prelevare il contenuto di un byte, manipolarlo per mezzo di operazioni e quindi reimmagazzinarlo nello stesso od in un altro posto. Per fare tutto ciò, il microprocessore dispone di particolari dispositivi chiamati "registri"

I registri

Esistono tre differenti registri, ognuno dei quali è predisposto a determinate funzioni, esattamente come un tennista che usa la mano destra per colpire la palla, la mano sinistra per lanciarla in aria prima di effettuare il servizio, oppure entrambe per allacciarsi le scarpe.
Questi registri hanno la facoltà di prelevare informazione dalle singole locazioni di memoria, anch'essi un solo valore per volta. È importante precisare come questi ultimi non rappresentino parte della memoria, in quanto non possiedono un indirizzo proprio; sono in effetti direttamente in relazione con il microprocessore, e separatamente gestibili grazie ad apposite istruzioni del linguaggio macchina.

L'accumulatore

Il primo registro di cui ci occuperemo è il registro A (meglio conosciuto sotto il nome di accumulatore). Come avrete modo di vedere nei capitoli seguenti, le funzioni svolte dall'accumulatore sono quelle a carattere più generale, fra queste citiamo la maggior parte delle operazioni aritmetiche del microprocessore.
Spesso e volentieri, il microprocessore ha necessità d'immagazzinare informazione in uno dei suoi registri prima di trattarla; l'istruzione (in forma mnemonica) che gli permette di prelevare il contenuto di una locazione di memoria (o più semplicemente un valore numerico compreso fra 0 e 255) ed inserirlo nell'accumulatore è LDA (LoaD Accumulator). Ad

esempio:

LDA 253

carica nell'accumulatore il contenuto della locazione di memoria posta all'indirizzo 253. L'equivalente in linguaggio macchina di questa istruzione è 165 253.
Notate come in quest'ultimo caso il codice macchina sia diviso in due parti, contrariamente ad RTS, identificato dal solo valore numerico 96. Il doppio valore è qui giustificato dal fatto che il primo di essi caratterizza unicamente l'istruzione vera e propria, LDA (165), mentre il secondo specifica l'indirizzo del byte contenente l'informazione da prelevare (253).
Queste due parti della medesima istruzione vengono quindi memorizzate in due distinte locazioni di memoria.

Gli indirizzamenti

La maggior parte delle istruzioni in linguaggio macchina possono esprimersi sotto varie forme, allo scopo di rendere più flessibile al programmatore la gestione della memoria, permettendogli di scegliere come e dove immagazzinare l'informazione trattata. La precedente istruzione LDA ne possiede otto differenti, chiamate più specificatamente "modi d'indirizzamento".
In base a particolari caratteristiche, questi modi d'indirizzamento alterano la tecnica attraverso la quale vengono specificati gli indirizzi delle locazioni interessate all'interno delle singole istruzioni.
Per fare un esempio, supponete di avere l'ordine di prelevare una lettera da una particolare casella postale; questo vostro compito può essere svolto in diversi modi:

[1] Vi è stato detto d'indirizzarvi alla casella 17.

[2] Vi è stato detto d'indirizzarvi alla terza casella da destra della seconda fila a partire dal basso.

[3] Vi è stato detto d'indirizzarvi alla casella di proprietà del signor Rossi.

[4] Vi è stato detto d'indirizzarvi alla casella il cui numero è a sua volta contenuto in un'altra casella.

[5] Vi è stato semplicemente detto di portare quella particolare lettera.

Troverete nei successivi capitoli ulteriori nozioni riguardanti i modi d'indirizzamento; per il momento ci limitiamo ad illustrarvene tre degli otto disponibili relativi all'istruzione LDA.

Modo 1: 165 253 (LDA 253)

Esso rappresenta una breve forma di LDA. Per ragioni che vi verranno spiegate in seguito, può accedere unicamente ad un piccolo intervallo d'indirizzi di memoria. Viene identificato attraverso il termine "indirizzamento in pagina zero".

Modo 2: 173 55 4 (LDA 1079)

Lunga forma di LDA, esso ha la facoltà di accedere a qualsiasi indirizzo di memoria. Notate come l'istruzione sia in questo caso suddivisa in tre distinte parti. La prima, 173, rappresenta LDA in questo particolare modo d'indirizzamento. Le due successive, 55 e 4, specificano l'indirizzo della locazione (1079) il cui contenuto deve essere immagazzinato nell'accumulatore; le ragioni per le quali il numero 1079 viene espresso in maniera così apparentemente inspiegabile vi verranno chiarite nel seguente capitolo. Tale modo è conosciuto sotto il nome di "indirizzamento assoluto"

Modo 3: 169 71 (LDA #71)

Differente dai due precedenti, questo modo non si occupa di prelevare il contenuto di una determinata locazione per immagazzinarlo quindi nell'accumulatore, ma provvede invece a porre in quest'ultimo un valore direttamente specificato. Nell'esempio considerato, tale valore è rappresentato dal numero 71, il quale non ha nulla a che vedere con l'indirizzo di memoria 71. Identificato attraverso il termine "indirizzamento immediato", esso viene caratterizzato in forma mnemonica dalla presenza del simbolo '#' alla sinistra del numero.

Abbiamo visto come far prelevare dal microprocessore una particolare informazione presente in memoria, ma prima di poter costruire qualcosa di utile, dobbiamo ancora imparare come fargliela manipolare. La funzione inversa di LDA viene svolta da STA (STore Accumulator), istruzione che provvede a scaricare in una specifica locazione di memoria il contenuto dell'accumulatore. STA dispone anch'essa di vari modi d'indirizzamento (sette in effetti), ma per il momento ci limitiamo a considerarne solamente due.

Modo 1: 133 41 (STA 41)

Questa istruzione scarica il contenuto dell'accumulatore nella locazione posta all'indirizzo di memoria 41. Analogamente ad LDA, tale forma in due parti (modo d'indirizzamento in pagina zero) può accedere soltanto

ad un ristretto intervallo di memoria (per l'esattezza le locazioni da 0 a 255).

Modo 2: 141 57 03 (STA 825)

Simile al modo 1, ma con la differenza che l'accesso è libero a tutte le 65536 locazioni di memoria del C16. Come per l'equivalente modo d'indirizzamento di LDA, il primo valore (141) rappresenta l'istruzione vera e propria, mentre i due successivi (57 e 3) costituiscono l'indirizzo della locazione 825.
Notate come non sia contemplato l'indirizzamento immediato per STA, il quale risulterebbe assolutamente privo di senso. Riprendendo il nostro precedente paragone, ciò equivarrebbe a scaricare la lettera non in una casella postale, ma direttamente nelle istruzioni che vi sono state date, il che risulta totalmente privo di significato.

Semplice routine di scrittura programmi

Scriveremo adesso alcuni programmini in linguaggio macchina allo scopo di esaminare le istruzioni appena trattate. Per rendere il lavoro più facile, digitate il seguente listato BASIC:

```
   5 PRINTCHR$(147);"......"
  10 REM QUESTO PROGRAMMA VI FACILITERA'
     L'INSERIMENTO DEI PROGRAMMI
      IN LINGUAGGIO MACCHINA
  20 READA
  30 IFA=-1THENGOTO70
  40 POKE8192+X,A
  50 X=X+1
  60 GOTO20
  70 PRINT"LOCAZIONE 3072 PRIMA...";PEEK(3072)
  80 SYS8192
  90 PRINT"LOCAZIONE 3072 DOPO...";PEEK(3072)
 100 END
1000 DATA169,1:REM LDA #1
1010 DATA141,0,12:REM STA 3072
1020 DATA96:REM RTS
9999 DATA-1
```

Linee 1000—9999:	contengono il programma in linguaggio macchina.
Linee 20—60:	memorizzano il nostro programma a partire dall'indirizzo di memoria 8192, in modo che possa essere eseguito.

Linee 70—90: mostrano il contenuto della locazione 3072 prima e dopo l'esecuzione del programma.

Ad esecuzione ultimata del BASIC, il nostro programma in linguaggio macchina risulterà immagazzinato in locazioni di memoria secondo il seguente formato:

INDIRIZZO	DATA
8192	169
8193	1
8194	141
8195	0
8196	12
8197	96

Per la gioia dei programmatori, eccovi anche la sua relativa forma mnemonica:

```
8192    LDA #1
8194    STA 3072
8197    RTS
```

Linguaggio assembly

Un programma redatto in forma mnemonica si dice scritto in "linguaggio assembly"; ciò in conseguenza del fatto che per trasformare la sequenza di caratteri comprensibile al programmatore nella sequenza di caratteri comprensibile al microprocessore, si utilizza un apposito programma denominato appunto "assemblatore". Nel corso del libro, eventuali programmi in linguaggio macchina vi verranno presentati in entrambi i formati:

INDIRIZZO	CODICE			ASSEMBLY
8192	169	1		LDA #1
8194	141	0	12	STA 3072
8197	96			RTS

Il precedente programma BASIC, oltre a caricare in memoria il nostro codice macchina, provvede anche a far partire la sua esecuzione (linea 80). Potrete agevolmente notare come, attraverso la nostra analisi del contenuto della locazione 3072 prima e dopo, il cambiamento ottenuto sia quello effettivamente auspicato. Il contenuto originale di tale locazione non ricopre alcun ruolo particolare, in quanto il programma lo altera immediatamente; è pertanto impossibile sapere cosa era presente in me-

moria prima di un vostro intervento modificatore. Il valore della locazione 3072 dopo l'esecuzione del programma è 1; questo dimostra a posteriori la corretta effettuazione delle istruzioni considerate (caricamento dell'accumulatore con il valore 1 e conseguente suo immagazzinamento in 3072).

Memoria di schermo

Vi è un risultato di questo programma molto probabilmente inatteso alla maggior parte di voi. Guardate nell'angolo in alto a sinistra dello schermo, e vedrete che conterrà una lettera 'A'. La linea 5 del BASIC si occupa di pulire il video, motivo per cui non può costituire un precedente residuo, e quindi è stata obbligatoriamente stampata dal programma in linguaggio macchina. Tutto quello che sappiamo è che quest'ultimo ha provveduto ad inserire il valore 1 nella locazione 3072; come si arriva dunque alla stampa della 'A' sullo schermo? Provate la stessa operazione in BASIC, ed osservate cosa succede. Premete CLR per ripulire il video, e digitate:

```
POKE 3072,1
```

Noterete la riapparizione della famosa 'A' nel medesimo angolo in alto a sinistra. Ciò è dovuto al fatto che la locazione 3072 ricopre una doppia funzione; viene infatti utilizzata per la visualizzazione di caratteri sullo schermo, oltre al normale immagazzinamento d'informazione. Il paragone con le caselle postali è sempre valido; esse dispongono quì di uno sportello trasparente che vi permette di vedere sul vostro schermo il suo contenuto interno. Riferendovi alla tabella dei codici di schermo nell'appendice [9], noterete come al valore 1 corrisponda effettivamente la lettera 'A' (il SET 1 viene impiegato inizialmente; per cambiare il set di caratteri, premete simultaneamente i tasti SHIFT e LOGO COMMODORE). Proviamo adesso a visualizzare altri caratteri. Per stampare ad esempio una 'X' dobbiamo innanzitutto consultare la precedente tabella in modo da trovare il suo corrispondente codice di schermo; noterete che questi equivale a 24. Per immagazzinarlo in memoria utilizzeremo il programma scritto in precedenza, ma questa volta cambiando LDA #1 in LDA #24. Servendoci del medesimo listato BASIC per la sua memorizzazione, è necessario alterare la linea 1000 in:

```
1000 DATA169,24:REM LDA #24
```

Il nostro programma in linguaggio macchina apparirà dunque come se-

gue (dopo l'esecuzione del BASIC):

8192	169	24		LDA #24
8194	141	0	12	STA 3072
8197	96			RTS

Facendolo partire, vedrete pertanto comparire una 'X' nel solito angolo in alto a sinistra dello schermo.
A questo punto vi domanderete come stampare qualcosa in altri punti dello schermo. La risposta è semplice: la cosiddetta "memoria di schermo" (l'insieme di queste caselle con lo sportello trasparente) si estende dalla locazione 3072 alla locazione 4071 compresa, corrispondente all'angolo in basso a destra. Essa viene suddivisa in 25 linee di 40 colonne, per un totale di 1000 locazioni di schermo. La numero 3073 si trova immediatamente alla destra della 3072, così come la (3072+40=3112) è situata al margine sinistro della seconda linea a partire dall'alto. Facendo uso del solito listato BASIC, proveremo ora a stampare una 'X' in quest'ultima locazione; il nostro programma diventa pertanto:

```
LDA #24 (lettera X)
STA 3112 (prima colonna seconda linea)
RTS
```

La linea 1010 del BASIC dovrà quindi leggersi:

```
1010 DATA141,40,12:REM STA 3112
```

Stampa di un messaggio

Faremo adesso uso del nostro listato BASIC per memorizzare un programma più grande che si occupi di stampare un messaggio sullo schermo. Dopo averlo caricato, aggiungetegli le linee seguenti:

```
1000 DATA169,8
1010 DATA141,0,12
1020 DATA169,5
1030 DATA141,1,12
1040 DATA169,12
1050 DATA141,2,12
1060 DATA141,3,12
1070 DATA169,15
1080 DATA141,4,12
1090 DATA96
```

Ad esecuzione avvenuta del BASIC, verrà visualizzato il messaggio HELLO nella prima linea di schermo a partire dal margine sinistro. Il programma in linguaggio macchina che ha generato questo messaggio si presenta così:

8192	169	8		LDA #8 (lettera H)
8194	141	0	12	STA 3072
8197	169	5		LDA #5 (lettera E)
8199	141	1	12	STA 3073
8202	169	12		LDA #12 (lettera L)
8204	141	2	12	STA 3074
8207	141	3	12	STA 3075
8210	169	15		LDA #15 (lettera O)
8212	141	4	12	STA 3076
8215	96			RTS

È interessante notare come per stampare la doppia 'L' non sia stato necessario ricaricare l'accumulatore con il valore 12 dopo il primo immagazzinamento; infatti, in qualsiasi trasferimento da memoria a registri o viceversa, il contenuto dell'origine viene automaticamente copiato senza subire alterazioni.
Possiamo scrivere gli stessi programmi appena visti impiegando diversi modi d'indirizzamento. Questa facoltà risulta particolarmente utile per aumentare la loro efficienza; a volte si desidera infatti privilegiare la rapidità esecutiva, altre la compattezza ed altre ancora la comprensibilità.
Modificheremo ora il precedente programma di stampa dei messaggi allo scopo di renderlo maggiormente versatile; aggiungete al suo corrispondente listato BASIC le seguenti linee:

```
  15 INPUT"CODICE DELLA LETTERA";B:POKE3,B
1000 DATA165,3:REM LDA 3
1090 DATA169,23:REM LDA #23
1100 DATA141,5,12:REM STA 3077
1110 DATA96:REM RTS
```

Il nuovo programma in linguaggio macchina apparirà nella seguente forma:

8192	165	3		LDA 3
8194	141	0	12	STA 3072
8197	169	5		LDA #5
8199	141	1	12	STA 3073
8202	169	12		LDA #12
8204	141	2	12	STA 3074

8207	141	3	12	STA 3075
8210	169	15		LDA #15
8212	141	4	12	STA 3076
8215	169	23		LDA #23
8217	141	5	12	STA 3077
8220	96			RTS

Notate come la prima lettera del messaggio venga prelevata dalla locazione numero 3 per mezzo di un indirizzamento in pagina zero, al posto del consueto indirizzamento immediato. La linea 15 del BASIC assegna a quest'ultima locazione un valore scelto da noi; provate ad eseguirlo varie volte rispondendo alla domanda con 25, 2 e 13.
Abbiamo visto in questo capitolo come la memoria possa ricoprire più di una funzione; è il caso delle locazioni dalla 3072 alla 4071 (memoria di schermo). Analogamente altre sue zone possono svolgere compiti speciali, come il controllo dei colori, della grafica, del suono, della tastiera, dei joysticks, e più generalmente gestione dell'input/output (comunicazioni con il mondo esterno). Queste particolari aree verranno citate nei successivi capitoli a titolo puramente introduttivo.

Riassunto del Capitolo 2

[1] Il microprocessore utilizza degli appositi registri per trasferimenti d'informazione e gestione generale della memoria.

[2] Esso dispone di tre specifici registri: 'X', 'Y' e l'accumulatore.

[3] Utilizziamo l'istruzione LDA per caricare un valore numerico nell'accumulatore.

[4] Utilizziamo l'istruzione STA per scaricare il valore contenuto nell'accumulatore in una particolare locazione di memoria.

[5] Queste istruzioni e molte altre ancora dispongono di vari modi d'indirizzamento che permettono una maggiore flessibilità di programmazione:
<*> L'indirizzamento immediato comprende il dato all'interno dell'istruzione.
<*> L'indirizzamento assoluto usa dati immagazzinati in una qualsiasi zona di memoria.
<*> L'indirizzamento in pagina zero usa dati posti unicamente nelle prime 256 locazioni di memoria.

[6] Un programma redatto in forma mnemonica si dice scritto in linguaggio Assembly.

[7] La memoria viene anche usata per la visualizzazione d'informazione sullo schermo.

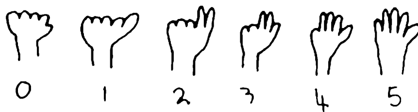
[8] Questa informazione viene mostrata in accordo ad un apposito codice secondo il quale ad ogni carattere stampabile è associato un valore numerico.

[9] Particolari zone di memoria vengono generalmente impiegate per la gestione delle funzioni di input/output (comunicazioni con il mondo esterno).

Introduzione all'esadecimale

Utilizzo dell'esadecimale

Nei precedenti capitoli avevamo parlato in più occasioni della memoria, ma senza tuttavia specificare il formato dell'informazione che ogni sua locazione può contenere. Ci eravamo soprattutto soffermati su come essa potesse immagazzinare numeri rappresentanti caratteri stampabili, istruzioni in linguaggio macchina ed indirizzi di memoria, limitandoci a descrivere le tecniche di trasferimento adottate dal microprocessore. È stato l'indirizzamento assoluto a mostrarci come il sistema di numerazione utilizzato dal computer non sia poi così semplice come si sarebbe potuto credere. Ad esempio, 141 5 12 equivale al codice macchina dell'istruzione STA 3077. Il primo valore, 141, rappresenta STA, mentre i successivi due, 5 e 12, caratterizzano l'indirizzo 3077; è perfettamente ovvio che a questo punto intervengono nuovi fattori non ancora presi in considerazione.
Proviamo a paragonare i registri del microprocessore a delle mani. Qual'è il numero più alto che le vostre mani possono contenere? Per rispondere a questa domanda bisogna innanzitutto chiarire che cosa significa "contenere". Potete usare le dita per contare fino a cinque, per cui una mano può contenere un numero compreso fra zero e cinque. Questo significa che il valore più alto esprimibile attraverso una mano è dunque cinque? Molti di voi rimarranno probabilmente sorpresi nel sapere che la risposta è NO. Contando da zero a cinque in questo modo



si limitano enormemente le capacità potenziali della mano, esattamente come l'adozione di un analogo sistema di numerazione per un computer.

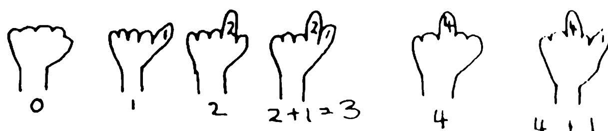
Sistema di numerazione binaria

Le "dita" di un computer possono essere alzate od abbassate (accese o spente) ma, come per le vostre, esso è inoltre in grado di precisare lo stato di ciascun dito. In altre parole, il valore rappresentato non dipende sol-

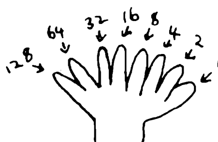
tanto dal numero di dita utilizzate, ma anche dalla loro rispettiva posizione. Sperimentatelo direttamente, assegnando ad ogni vostro dito uno dei seguenti valori (scriveteli con una penna se vi piace).



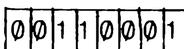
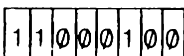
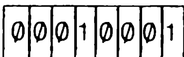
Provate adesso a contare addizionando i numeri rappresentati da ciascun dito alzato.



Cercate di rappresentare per mezzo di questa tecnica i numeri 7, 16, 10, 21, 29.
Ci domandiamo ora qual'è il più alto valore ottenibile con cinque dita. La risposta è semplice: 1+2+4+8+16 = 31.
Vi renderete immediatamente conto di come trentuno costituisca un sensibile miglioramento del limite originariamente fissato a cinque. Le mani del computer si differenziano dalle nostre sotto vari aspetti. Le sue dita sono rappresentate da segnalazioni elettroniche che possono essere interpretate come accese o spente (così come le vere dita possono essere alzate od abbassate). Per il beneficio dei programmatori, la condizione di accensione viene caratterizzata dal valore 1, mentre quella di spegnimento dal valore 0. L'altra principale differenza è costituita dal fatto che il computer possiede otto "dita" per ogni "mano". Ciò potrebbe sembrarvi stupido, ma non vi è nessuna ragione che possa confermare una tale impressione. Questa speciale mano di otto dita viene identificata attraverso il termine "byte", sinonimo di locazione di memoria. Riprendendo il precedente esempio, assegneremo a ciascun "dito" del byte uno dei seguenti valori: 1, 2, 4, 8, 16, 32, 64, 128.



Anche in questo caso procederemo a contare addizionando i valori associati ad ogni singolo dito alzato.

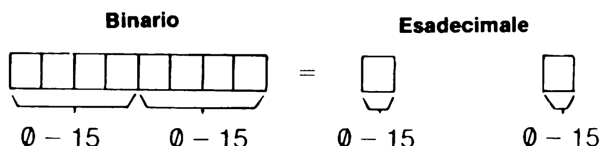
Mano di otto dita	'Mano' del computer (byte)	Numero
		32+16+1 = 49
		128+64+4 = 196
		16+1 = 17

Qual'è allora il massimo valore rappresentabile con la "mano ad otto dita" del computer? Il calcolo è presto fatto: 1+2+4+8+16+32+64+128 = 255 Senza rendercene conto, in questo capitolo abbiamo presentato il sistema di numerazione binaria (base due). Tutti i computers operano in base due, rappresentando accensioni e spegnimenti elettrici sotto forma di un flusso interminabile di zero e uno. Questo certamente rende maggiormente difficoltosa al programmatore la comprensione dell'attività interna del suo calcolatore. Vediamo infine un breve esempio riepilogativo:

ASSEMBLY	CODICE MACCHINA			CONFIGURAZIONE BINARIA
LDA #8	169	8		10101001 00001000
STA 3077	141	5	12	10001101 00000101 00001100
RTS	96			01100000

Perché l'esadecimale

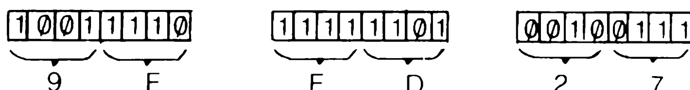
Numeri in configurazione binaria sono certamente impossibili da ricordare per un programmatore, ed inoltre difficili da digitare correttamente. Si potrebbe utilizzare il sistema decimale come presentato nella colonna del codice macchina, ma ben presto ci si renderebbe conto di quanto non risulti appropriato; ci affideremo dunque all'esadecimale, o base sedici. Vi sembrerà forse anomala una base superiore a dieci ma vi assicuriamo che, contrariamente alle apparenze, questa è molto facile da gestire in quanto è strettamente in relazione con l'effettiva struttura binaria adottata dal computer. Le conversioni fra binario ed esadecimale sono estremamente semplici. Ogni cifra esadecimale può infatti contenere valori compresi tra zero e quindici, esattamente come ogni cifra decimale può variare da un minimo di zero ad un massimo di nove. Si nota dunque come una singola cifra esadecimale rappresenti la metà di un byte.



L'intero byte può quindi essere caratterizzato da due sole cifre in base sedici, invece di otto in base due. Vi sarete probabilmente chiesti in che modo una singola cifra possa variare da zero a quindici; tutto si svolge esattamente come in base dieci, ma con i numeri 10, 11, 12, 13, 14 e 15 rispettivamente rappresentati dalle lettere A, B, C, D, E ed F.

BINARIO	DECIMALE	ESADECIMALE
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
10000	16	10

Lo schema seguente vi mostra come la conversione da binario ad esadecimale si riduca alla suddivisione del byte in due distinti segmenti di quattro "dita" ognuno.



Struttura matematica delle basi

Matematicamente ogni base, sia essa dieci, due, sedici o centosettantanove, segue un semplice formato. Ogni cifra assume infatti il valore V*(BASE)↑Posizione−1.
In altre parole, il numero decimale 98617 equivale a:

7*[10↑0]+1*[10↑1]+6*[10↑2]+8*[10↑3]+9*[10↑4] = 98617
(7*1)+(1*10)+(6*100)+(8*1000)+(9*10000) = 98617
7 + 10 + 600 + 8000 + 90000 = 98617

In binario, 01011101 equivale a:

1*2↑0+0*2↑1+1*2↑2+1*2↑3+1*2↑4+0*2↑5+1*2↑6+0*2↑7 = 93
1*1+0*2+1*4+1*8+1*16+0*32+1*64+0*128 = 93
1 + 0 + 4 + 8 + 16 + 0 + 64 + 0 = 93

In esadecimale, A7C4E equivale a:

14*[16↑0]+4*[16↑1]+12*[16↑2]+7*[16↑3]+10*[16↑4] = 687182
(14*1)+(4*16)+(12*256)+(7*4096)+(10*65536) = 687182
14 + 64 + 3072 + 28672 + 655360 = 687182

È necessario fare alcune precisazioni. Innanzitutto, ogni numero immagazzinabile in una locazione di memoria (compreso quindi fra 0 e 255) può essere rappresentato attraverso otto cifre binarie (bits) corrispondenti alle famose pseudo—dita del computer, oppure attraverso due cifre esadecimali (FF = 15*16+15 = 255). Questa limitazione costituisce il problema precedentemente riscontrato a proposito dell'indirizzamento assoluto; se non è possibile inserire in un byte numeri superiori a 255, come si fa allora a specificare indirizzi di memoria compresi fra 0 e 65535 (64 Kbytes)? La soluzione risiede nell'utilizzare due locazioni contigue, non da addizionare fra loro ma da gestire come parti integranti del medesimo valore; parlando di indirizzamento assoluto, si considerano automaticamente numeri binari formati da sedici bits (due bytes). In base a quanto visto in precedenza, si può dunque associare un indirizzo di memoria a quattro cifre esadecimali. Il massimo valore esprimibile con tale numero di cifre è:

FFFF = (15*1) + (15*16) + (15*256) + (15*4096)
= 15 + 240 + 3840 + 61440
= 65535 = 64 Kbytes

Questo valore è sufficientemente grande per indirizzare l'intera memoria disponibile.

Indirizzamento assoluto

Rileggendo l'inizio del capitolo, noterete l'esistenza di un problema riguardante l'impiego dell'indirizzamento assoluto che ci siamo appena sforzati di risolvere. Un'altra caratteristica da ricordarsi assolutamente utilizzando tale modo d'indirizzamento è che i due bytes formanti l'indirizzo di memoria interessato sono SEMPRE memorizzati in senso inverso. ad esempio:

1029 (decimale) equivale a 0405 (esadecimale)
LDA 1029 viene immagazzinato in (141) (05) (04)

Il byte più significativo, o byte alto (04) viene posizionato per ultimo, preceduto dal byte meno significativo, o byte basso (05) Notate come questa convenzione sia opposta al modo normale di scrittura dei numeri; effettivamente, per fare un esempio, la cifra più significativa di 17 è 1 (1*10) e si trova alla sinistra di 7 (7*1), cifra meno significativa Per particolari ragioni non facilmente precisabili, i bytes di un indirizzo assoluto sono sempre memorizzati nel formato byte basso/byte alto
Vediamo adesso le limitazioni relative all'indirizzamento in pagina zero. Tutte le istruzioni formate da due bytes lasciano soltanto uno di questi a disposizione per specificare l'indirizzo. Avevamo precedentemente dimostrato come il massimo valore memorizzabile in una locazione od un registro sia 255 (in base 10); la cosiddetta pagina zero è dunque composta dalle prime 256 locazioni di memoria del computer, ovvero quelle numerate da 0 a 255. Più generalmente, un blocco di 256 bytes viene chiamato "pagina"
Per distinguere i numeri decimali da quelli esadecimali, nei seguenti capitoli adotteremo la convenzione standard di far precedere questi ultimi dal segno '$'.

```
LDA 3072   equivale a LDA $0C00
LDA 65535  equivale a LDA $FFFF
LDA 0      equivale a LDA $0
```

D'ora in poi, tutti i listati dei programmi in linguaggio macchina adotteranno anch'essi la notazione esadecimale.

```
2000    A9 08       LDA #$08
2002    8D 00 0C    STA $0C00
2005    A9 53       LDA #$53
2007    8D 01 0C    STA $0C01
200A    60          RTS
```

Conversione da esadecimale a decimale

Abbiamo provveduto ad inserire nell'appendice [3] un'apposita tavola per rapide conversioni da esadecimale a decimale. Utilizzando questa tavola con numeri formati da un singolo byte, consultate le colonne verticali per la prima cifra esadecimale e le linee orizzontali per la seconda. Vediamo un esempio:

$2A (terza linea undicesima colonna)
La relativa casella contiene i seguenti valori:
42 (BYTE BASSO) 10752 (BYTE ALTO)

Il valore espresso come byte basso (42) corrisponde a $2A esadecimale. Nel caso di numeri formati da due bytes, questi vanno inizialmente scomposti in byte basso e byte alto, e la loro notazione decimale viene quindi ottenuta sommando i due rispettivi equivalenti dati dalla tavola.

$7156 ($71 byte alto e $56 byte basso)
<*> BYTE ALTO ($71) in ottava linea e seconda colonna.
Valore per byte alto: 28928
<*> BYTE BASSO ($56) in sesta linea e settima colonna.
Valore per byte basso: 86
$7156 = (28928+86) = 29014

Notate che in qualunque caso, il valore dato per il byte alto corrisponde a duecentocinquantasei volte quello dato per il byte basso (in effetti, 42*256 = 10752).

Riassunto del Capitolo 3

[1] Contando sulle "dita" del computer, la loro posizione è altrettanto importante come il loro numero.

[2] Ogni registro ed ogni locazione di memoria hanno otto "dita" (bits), ed il massimo valore che sono in grado di contenere è 255.

[3] Una locazione di memoria composta da otto "dita" è denominata "byte"

[4] Ogni "dito" ha un suo proprio valore in funzione della posizione occupata all'interno del byte.

[5] Esadecimale (base sedici) costituisce una forma più sintetica della notazione binaria. Quattro cifre binarie corrispondono ad una cifra esadecimale.

[6]	DECIMALE:	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
	ESADECIMALE:	0 1 2 3 4 5 6 7 8 9 A B C D E F 10

[7] L'indirizzamento in pagina zero può accedere soltanto alle prime 256 locazioni di memoria, ovvero il numero massimo indirizzabile con un byte.

[8] L'indirizzamento assoluto può accedere a qualunque locazione di memoria compresa fra le 65536 (64 Kbytes) disponibili, ovvero il numero massimo indirizzabile per mezzo di due bytes.

[9] Gli indirizzi assoluti sono SEMPRE memorizzati nel formato byte basso/byte alto (8D 98 17 LDA $1798).

[10] I numeri esadecimali sono preceduti dal simbolo '$'.

[11] Ricordatevi di utilizzare la tavola di conversione da esadecimale a decimale posta nell'appendice [3].

Introduzione a Tedmon

Tedmon è un monitor per linguaggio macchina, nonchè mini assemblatore e disassemblatore, che risiede nella memoria ROM del C16, e pertanto risulta incorporato nel computer. Viene utilizzato come aiuto per l'inserimento e la correzione dei programmi in linguaggio macchina, ed inoltre permette di esaminare direttamente qualsiasi zona di memoria scelta dal programmatore.
Analogamente al BASIC, Tedmon dispone di una serie di comandi basati su una precisa e rigorosa sintassi. Ogni volta che deciderete di usarlo, ricordatevi sempre d'impiegare tali comandi nel modo corretto.

<A> **Assemblaggio**

Assembla una linea di linguaggio macchina 6502/7501. Questo comando facilita notevolmente l'inserimento dei codici mnemonici. Potendo assemblare una sola linea alla volta, non contempla l'impiego di etichette. Deve utilizzarsi nel seguente formato:

.A (indirizzo) (mnemonico) (operando)

Eccovi un esempio pratico:

.A 1000 LDA #$08

Premendo il tasto <RETURN>, questa linea viene espansa in modo da includervi i valori esadecimali corrispondenti al codice mnemonico dell'istruzione assemblata, nonchè ai suoi eventuali parametri. Riprendendo l'esempio precedente, il risultato è il seguente:

.A 1000 A9 08   LDA #$08

Dopo che una linea di linguaggio macchina è stata positivamente assemblata, il monitor visualizza automaticamente il successivo indirizzo utilizzabile.
Qualora si desideri uscire da questo comando, è sufficiente premere il tasto <RETURN>. Nel caso il monitor scopra un eventuale errore di sintassi, provvederà a segnalarlo immediatamente stampando un punto interrogativo al termine della linea.

<C> Comparazione di aree di memoria

Questo comando compara due distinte aree di memoria, provvedendo a segnalare le locazioni aventi un contenuto differente. Viene usato nel modo seguente:

.C (inizio) (fine) (inizio seconda area)

Vediamo un esempio pratico:

.C 2000 2FFF 3000

Con questo comando ordiniamo a Tedmon di confrontare il contenuto delle locazioni da $2000 a $2FFF con quello delle locazioni da $3000 a $3FFF. Eventuali differenze fra le due aree verranno segnalate attraverso la stampa degli indirizzi relativi al primo intervallo.

<D> Disassemblaggio

Provvede a disassemblare una zona della memoria. Deve essere utilizzato nel seguente formato:

.D (indirizzo iniziale) (indirizzo finale facoltativo)

Eccovi due esempi relativi ad ogni sintassi consentita:

.D 1000

od in alternativa:

.D 1000 3000

Nel primo caso verrà mostrato il disassemblato delle prime dieci linee di codice macchina a partire dall'indirizzo specificato; per visualizzare le dieci linee successive, premere nuovamente <D> e <RETURN>.
Il secondo esempio disassemblerà invece tutto il codice macchina compreso fra i due indirizzi specificati (nel nostro caso, $1000 e $3000).
Non spaventatevi se, disassemblando una particolare area di memoria, notate in uscita una specie di "sporcizia" indecifrabile del tipo:

```
.1000  02         ???
.1001  AF         ???
.1002  20 02 AF   JSR F02
.1005  02         ???
```

La spiegazione è semplice: quella determinata zona non contiene vere e proprie istruzioni di linguaggio macchina, ma molto probabilmente semplici dati numerici, relativi ad esempio a grafica in alta risoluzione (??? indica un valore non riconosciuto da Tedmon come istruzione effettivamente implementata nel 6502/7501).

<F> **Riempimento di aree di memoria**

Questo comando riempie una determinata area di memoria con uno specificato valore numerico esadecimale compreso nell'intervallo $00—$FF— La sua sintassi è:

.F (indirizzo iniziale) (indirizzo finale) (valore)

Esempio:

.F 1000 4000 00

Il precedente comando riempie l'area di memoria da $1000 a $4000 con il valore $00.

<G> **Esecuzione di programmi in linguaggio macchina**

Provoca l'esecuzione di un programma in linguaggio macchina a partire da uno specificato indirizzo di memoria, oppure nel caso quest'ultimo non sia stato precisato, a partire dal corrente valore del program counter (contatore di programma del microprocessore). Adotta il seguente formato:

.G (indirizzo di partenza facoltativo)

Esempio:

.G 2FE0

Con questo comando si provocherà l'esecuzione di un programma in codice macchina a partire dall'indirizzo $2FE0. Verificate sempre l'effettiva presenza di un programma prima di utilizzare questo comando; saltando ad una locazione contenente "sporcizia", si potrebbe infatti ricadere in una situazione di crash (blocco del sistema), con la spiacevole conseguenza di dover spegnere e riaccendere il computer per sbloccarlo.

<H> **Ricerca di valori in memoria**

L'impiego di questo comando permette la ricerca di particolari valori esadecimali all'interno di una determinata zona di memoria. Esso rappresenta indubbiamente una delle più utili e potenti facilitazioni messe a disposizione da Tedmon. La sua sintassi è:

.H (indirizzo iniziale) (indirizzo finale) (valore) (...)

Esempio:

.H 1000 2000 20 37 FD

Il precedente comando ricerca nell'intervallo di memoria $1000—$2000 la sequenza di bytes $20—$37—$FD e mostra l'indirizzo del primo valore in caso di riscontro.
Esso può inoltre venire impiegato per ricercare stringhe di caratteri ASCII, semplicemente specificando la stringa desiderata racchiusa fra apostrofi:

.H 3000 4000 'HELLO'

L'esempio qui sopra provvede a ricercare la presenza della stringa "HELLO" all'interno dell'intervallo $3000—$4000, ed in caso positivo mostra l'indirizzo (o gli indirizzi se riscontrata più volte) del carattere "H"
Questo comando risulta infine estremamente pratico per ricercare determinate istruzioni in linguaggio macchina, previa loro traduzione in sequenze di valori esadecimali.

<L> **Caricamento di zone di memoria dalle periferiche**

Carica in memoria un programma prelevandolo da nastro o da disco. Si comporta esattamente come il comando LOAD del BASIC, ma con la differenza che il caricamento avviene sempre in formato non rilocabile nella stessa area di memoria dalla quale era stato salvato. Vi consigliamo pertanto di limitarne l'impiego ai soli programmi in linguaggio macchina. Adotta la seguente sintassi:

.L "NOME DEL PROGRAMMA",(canale del dispositivo usato)

Un paio di esempi potrebbero essere:

.L "TOPOLINO",08 (da disco)
.L "PAPERINO",01 (da nastro)

<M> **Visualizzazione del contenuto della memoria**

Permette di esaminare un'area di memoria tanto sotto forma di contenuti numerici quanto di caratteri ASCII. Insieme al comando di disassemblaggio, è senza dubbio uno dei più frequentemente utilizzati dai programmatori. La sua sintassi è la seguente:

.M (indirizzo iniziale) (indirizzo finale facoltativo)

Esempio:

.M 1000

oppure

.M 1000 2000

Nel primo caso verranno visualizzate dodici linee di otto bytes ciascuna a partire dall'indirizzo $1000, mentre nel secondo caso verranno mostrati i contenuti di tutte le locazioni comprese nell'intervallo $1000–$2000. Un esempio di ciò che si presenta in uscita è il seguente:

```
.M 1000 1008
    1000 01 02 03 04 05 06 07 08  ........
    1008 41 42 43 44 45 46 47 48 ABCDEFGH
```

Alla destra degli otto valori esadecimali di ogni riga sono presenti i loro rispettivi caratteri ASCII (in negativo). Nel caso ad un particolare valore non corrisponda nessun carattere, Tedmon provvederà ad indicarlo per mezzo di un punto (sempre in negativo).

<R> **Visualizzazione dei registri del microprocessore**

Questo comando permette di esaminare il contenuto dei vari registri del microprocessore 7501. La sintassi è:

.R

Il risultato ottenuto sullo schermo è il seguente:

```
     PC    SR    AC    XR    YR    SP
;   0000   00    00    00    00    F8
```

[PC] = program counter (contatore di programma)
[SR] = registro di stato
[AC] = accumulatore

[XR] = registro X
[YR] = registro Y
[SP] = stack pointer (puntatore dello stack)

È necessario puntualizzare come i precedenti valori non corrispondano a quelli effettivamente immagazzinati nei vari registri; dopotutto, il monitor stesso non è altro che un grosso programma in linguaggio macchina, e pertanto altera continuamente il contenuto di tali registri.

<S> Salvataggio di zone di memoria

Provvede a registrare su disco o nastro una determinata zona di memoria sotto forma di programma. Il suo funzionamento è simile al comando SAVE del BASIC, ma tuttavia molto più flessibile in quanto permette di specificare gli indirizzi d'inizio e di fine del blocco da salvare. Si esprime attraverso la seguente sintassi:

```
.S "NOME DEL PROGRAMMA",(dispositivo),(inizio),(fine)
```

Vediamo un paio di esempi:

```
.S "TOPOLINO",08,0C00,0A00 (da disco)
.S "PAPERINO",01,0C00,0A00 (da nastro)
```

<T> Trasferimento di blocchi di memoria

Permette di spostare in altra posizione un determinato blocco di locazioni di memoria. Risulta estremamente utile per copiare automaticamente dati ricorrenti senza doverli ribattere, oppure per generare un duplicato interno di particolari routines in linguaggio macchina. Adotta la seguente sintassi:

```
.T (inizio) (fine) (indirizzo iniziale nuova zona)
```

Esempio:

```
.T 1000 1FFF 3000
```

Con questo comando vengono ricopiati i contenuti delle locazioni \$1000—\$1FFF nel nuovo intervallo \$3000—\$3FFF.

<V> Verifica di zone di memoria registrate

Provvede a verificare che un programma registrato su disco o nastro sia esattamente identico a quanto presente in memoria. Equivale sostanzial-

mente al comando VERIFY del BASIC. Nel caso riscontri eventuali differenze, le segnalerà per mezzo di una condizione di errore. Estremamente utile per assicurarsi che un programma venga salvato correttamente. La sua sintassi:

.V "NOME DEL PROGRAMMA",(canale del dispositivo usato)

Eccovi due esempi:

.V "TOPOLINO",08 (da disco)
.V "PAPERINO",01 (da nastro)

<X> Ritorno al BASIC

Con questo comando si uscirà da Tedmon, restituendo il controllo all'interprete BASIC.

<.> Assemblaggio

Assembla una linea di linguaggio macchina 6502/7501 è identico al comando <A> (vedere).

<>> Modifica di locazione di memoria

Permette di modificare il contenuto di locazioni di memoria di indirizzo specificato (vedere il comando <M>).

<;> Modifica del contenuto dei registri

Permette di aggiornare i valori immagazzinati nei registri del microprocessore 7501 prima che venga dato un eventuale comando <G>.

Per entrare in Tedmon dal BASIC, digitate il comando MONITOR seguito dal tasto <RETURN>. Vedrete quindi il seguente schema apparire sul video:

PC	SR	AC	XR	YR	SP
0000	00	0	00	00	F8

Esso rappresenta l'attuale contenuto di ogni singolo registro del microprocessore 7501. Il cursore lampeggiante indica che Tedmon è a questo punto in attesa di ricevere un vostro comando.

Vi suggeriamo di familiarizzare con Tedmon provando a digitare alcuni

degli esempi appena presentati, oppure qualsiasi altro di vostra elaborazione. Questo vi permetterà di prendere sempre maggiore confidenza con la rigorosa sintassi dei suoi comandi. Vi renderete ben presto conto di come Tedmon rappresenti un formidabile aiuto nello sviluppo di programmi in linguaggio macchina per il vostro C16.

CAPITOLO 5

Dotazione del microprocessore

Nei quattro capitoli precedenti abbiamo trattato le basi principali riguardanti le innumerevoli complicazioni proprie del linguaggio macchina. Da ora in poi ci addentreremo nell'argomento sempre più dettagliatamente. Siamo sufficientemente preparati per andare adesso ad occuparci delle applicazioni aritmetiche inerenti all'impiego del codice macchina.

Immagazzinamento dei numeri

Abbiamo appreso dal terzo capitolo che il maggior numero immagazzinabile in un byte (locazione di memoria) è 255. Avevamo inoltre visto come per indirizzi superiori a tale valore fossero necessari due bytes per rappresentarli nel formato byte basso/byte alto.

Indirizzo = byte basso + 256 * byte alto

Basandoci sul medesimo principio possiamo dunque rappresentare qualsiasi numero maggiore di 255 e minore di 65536 (65535 = 255 + 256 * 255), ed anche numeri più grandi purchè si faccia uso di almeno un terzo byte.

Numero = byte 1 + (256 * byte 2) + (65536 * byte 3) +

Il flag di carry

Addizionando fra loro due numeri di un solo byte, è tuttavia possibile che la somma risulti maggiore di 255. Come possiamo dunque esprimere un tale risultato? Se lo ponessimo in un unico byte non potrebbe eccedere 255, per cui

207 + 194 = 401 modulo 255 = 145

ma anche

58 + 87 = 145

Ovviamente qualcosa non quadra. In un modo o nell'altro, deve comunque risultare possibile immagazzinare l'informazione supplementare relativa ad una somma maggiore di 255. Per questo è stato provvisto il

microprocessore 7501 di un apposito bit di segnalazione denominato "flag di carry" (riporto). Il flag di carry viene posto a uno nel caso di risultati maggiori di 255.

207 + 194 = 145 (carry 1)
 58 + 87 = 145 (carry 0)

Un singolo bit è sufficiente per ricoprire tutti i casi possibili di riporto.

```
    11111111              255
  + 11111111            + 255
  1 11111110 (carry)    = 254 + carry
```

Per addizionare fra loro due numeri a due bytes, bisogna innanzitutto sommare i rispettivi bytes bassi, immagazzinare il risultato e quindi addizionare i bytes alti includendo l'eventuale carry della prima operazione. Vediamo ad esempio come sommare $30A7 con $2CC4:

```
BYTES BASSI              BYTES ALTI
   A7                      30
 + C4                    + 2C
 = 6B  (carry = 1)       + 1  (carry)
                         = 5D
```

Risultato finale: $30A7 + $2CC4 = $5D6B

Somma di numeri

L'istruzione in linguaggio macchina che si occupa di sommare fra loro due numeri di un solo byte è ADC (ADd with Carry). Essa provvede a sommare il valore specificato (od il contenuto di un byte) con l'accumulatore, lasciando in quest'ultimo il risultato. L'eventuale accensione del flag di carry viene automaticamente eseguita se necessario. Non potendo determinare lo stato anteriore di tale flag, è indispensabile azzerarlo prima di effettuare un'addizione che non comprenda precedenti riporti. A questo scopo è prevista l'istruzione CLC (CLear Carry). Digitate il seguente programma per mezzo di Tedmon:

```
2000 LDA #$00
2002 STA $05
2004 LDA #$03
2006 CLC
2007 ADC #$05
2009 STA $05
200B BRK
```

Sempre attraverso il monitor, battete .G 2000, quindi .M 0005. La locazione di memoria $05 dovrebbe contenere il valore $08. Modificheremo adesso il programma per alterare la somma da esso effettuata. Digitate:

```
2000 LDA #$00
2002 STA $05
2004 LDA #$27
2006 CLC
2007 ADC #
2009 STA $05
200B BRK
```

Battete .G 2000 per eseguire il programma, quindi .M 0005. Il valore contenuto nella locazione $05 sarà ora $1B. In effetti, + $27 = $11B, per cui possiamo affermare che il carry è stato posto a uno. A questo punto, modificheremo nuovamente il programma, accendendo deliberatamente il carry per mezzo dell'istruzione SEC (SEt Carry) prima di eseguire la nostra addizione. Digitate le linee seguenti:

```
2000 LDA #$03
2002 SEC
2003 ADC #$05
2005 STA $05
2007 BRK
```

Eseguite il programma con .G 2000, quindi battete .M 0005. Noterete che la locazione $05 conterrà il valore $09, risultato della seguente operazione:

```
  3
+ 5
+ 1 (carry)
---
= 9
```

A partire da questi esempi, possiamo vedere come il flag di carry venga riportato dal risultato di un'addizione ad una successiva operazione. Utilizzeremo adesso questa tecnica per l'addizione con numeri di due bytes.

Addizione con numeri di due bytes

Supponiamo di voler addizionare i numeri $6C67 e $49B2. Per fare questo dobbiamo scomporre l'operazione in due addizioni semplici di numeri di un solo byte.

BYTES BASSI		BYTES ALTI	
67		6C	
+ B2		+ 49	
= 19	(carry = 1)	+ 1	(carry)
		= B6	

Digitate il seguente programma:

```
2000  LDA #$67
2002  CLC
2003  ADC #$B2
2005  STA $03
2007  LDA #$6C
2009  ADC #$49
200B  STA $04
200D  BRK
```

Al termine della sua esecuzione avremo il byte basso ed il byte alto della somma rispettivamente immagazzinati nelle locazioni $03 (19) e $04 (B6):

Risultato finale: $6C67 + $49B2 = $B619

Sottrazione di numeri

La precedente procedura può essere inoltre estesa a numeri formati da più di due bytes. Il microprocessore, così come dispone di un'istruzione di addizione, ne possiede anche una di sottrazione. Similmente ad ADC, SBC (SuBtract with Carry) utilizza il flag di carry nei suoi calcoli. A causa del modo nel quale il microprocessore effettua le sottrazioni, tale flag viene invertito nel corso dell'operazione. Vediamo un esempio:

8		8
− 5		− 5
− 1	ma	− carry (carry = 1)
= 2		= 3

Conseguentemente, per eseguire una sottrazione senza riporto, il flag di carry deve essere posto a uno prima di usare l'istruzione SBC. Digitate quanto segue:

```
2000 LDA #$08
2002 CLC
2003 SBC #$05
2005 STA $05
2007 BRK
```

Eseguitelo con .G 2000, quindi esaminate il risultato ottenuto per mezzo di .M 0005. Noterete che, avendo azzerato il carry invece di porlo a uno, il risultato non corrisponde alla realtà. Correggeremo il nostro errore cambiando il CLC in $2002 con SEC. Rieseguite il programma, e controllate nuovamente il contenuto della locazione $05. Vedrete che adesso il risultato sarà finalmente esatto.

```
  8                      8
– 5                    – 5
– 1  (carry = 0)       – 0  (carry = 1)
———                    ———
= 2                    = 3
```

Vi sarete probabilmente chiesti come il microprocessore esegua le sottrazioni aventi un risultato minore di zero. Provate ad esempio con 8 – E = –6; modificate l'istruzione SBC #$05 in SBC #$0E e rieseguite il programma.

```
  8                          108  (carry = 0)
– E                          – E
———        oppure:           ———
– 6                          FA
```

Notate che (–6 = 0 – 6 = FA) e che (FA + 6 = 0).
Questa tecnica di azzeramento del carry indicante un "prestito" di 256 può essere utilizzata per sottrazioni di numeri a più bytes, analogamente alle corrispondenti addizioni. Provate a scrivere ora un programma che esegua la seguente sottrazione: $E615 – $7198

Eccovi un esempio:

```
2000  LDA #$15
2002  SEC
2003  SBC #$98
2005  STA $03
2007  LDA #$E6
2009  SBC #$71
200B  STA $04
200D  BRK
```

Digitatelo, eseguitelo, e prendete nota dei risultati. Combinando i bytes alto e basso contenuti rispettivamente in $04 e $03, otterrete il valore $747D.
Le istruzioni ADC e SBC possono essere utilizzate in vari modi d'indiriz-

zamento, come la maggior parte delle altre. In questo capitolo ci siamo limitati ad esprimerle in indirizzamento immediato.
Notate che SEC e CLC dispongono soltanto di un unico modo d'indirizzamento, denominato "implicito". Esse eseguono uno specifico compito all'interno di uno specifico registro, per cui non possiedono nessun indirizzamento alternativo. Il loro modo d'indirizzamento è infatti implicito nell'istruzione stessa.

Esercizio

Scrivete un programma per addizionare il valore $37 al contenuto della locazione di memoria $05 utilizzando ADC in indirizzamento assoluto, e quindi immagazzinate nella stessa il risultato finale.
Considerate che:

```
LDA #$FF
CLC
ADC #$01
```

lascia il valore $00 nell'accumulatore mettendo contemporaneamente il flag di carry a uno, e che:

```
LDA #$00
SEC
SBC #$01
```

lascia il valore $FF nell'accumulatore azzerando nello stesso tempo il flag di carry (ad indicare un "prestito"). In quest'ultimo caso abbiamo quello che si chiama "rientro"; contando in ordine crescente oltre 255 si riprende dallo zero, mentre contando in senso inverso oltre lo zero si riprende da 255 in giù.

Riassunto del Capitolo 5

[1] È possibile rappresentare attraverso più di un byte numeri di qualsiasi grandezza.
Num = byte1 + (256 * byte2) + (65536 * byte3) +

[2] Il microprocessore 7501 possiede un flag di carry che viene posto a uno per indicare un precedente riporto nelle somme di numeri di due bytes.

[3] ADC provvede a sommare due bytes più il contenuto del flag di carry. Una CLC deve essere impiegata nel caso il carry non intervenga nell'addizione.

[4] ADC pone a uno il flag di carry nel caso il risultato sia maggiore di 255, altrimenti lo azzera. Il nuovo contenuto dell'accumulatore è sempre minore di 256.

[5] SBC provvede a sottrarre un determinato valore da quello contenuto nell'accumulatore, per poi sottrarre anche l'inverso del flag di carry. Per evitare che quest'ultimo possa interferire con il calcolo, una SEC deve essere impiegata prima della SBC.

[6] SBC pone a uno il flag di carry nel caso il risultato non richieda un "prestito" (A − M >= 0), altrimenti lo azzera. Il risultato lasciato nell'accumulatore è allora 256 − (A − M).

[7] Addizione di numeri di due bytes.
Azzerare il carry
XX = Somma dei bytes bassi + (carry = 0)
YY = Somma dei bytes alti + (carry = ?)
Risultato = $YYXX

[8] Sottrazione di numeri di due bytes.
Mettere a uno il carry
XX = Differenza dei bytes bassi − inverso (carry = 1)
YY = Differenza dei bytes alti − inverso (carry = ?)
Risultato = $YYXX

CAPITOLO 6

Controllo del programma

Iterazione utilizzante JMP

L'istruzione di salto incondizionato del linguaggio macchina è JMP (JuMP). Esattamente come per il comando GOTO del BASIC, è necessario segnalare anche la destinazione del salto; nel nostro caso, si tratterà di un indirizzo di memoria strutturato nel classico formato byte basso/byte alto (indirizzamento assoluto).
Utilizzeremo questa istruzione per costruire un programma equivalente al seguente listato BASIC:

```
100 X=X+4
110 GOTO 100
```

Per darvi un'idea di cosa succeda effettivamente durante l'esecuzione del programma, addizioneremo il valore $04 al contenuto della locazioe $0C00 della memoria di schermo. Digitate quanto segue per mezzo di Tedmon:

```
2000  LDA #$00
2002  STA $0C00
2005  LDA $0C00
2008  CLC
2009  ADC #$04
200B  STA $0C00
200E  JMP $2005
```

Eseguitelo con .G 2000, ed osservate quanto rapidamente vengano visualizzati i caratteri all'interno del quadratino lampeggiante situato nell'angolo in alto a sinistra dello schermo.

Cicli infiniti

Avrete senz'altro notato come il programma precedente stia ancora proseguendo nell'esecuzione. Così come il BASIC:

```
100 X=X+4
110 GOTO 100
```

il nostro programma continua a girare perennemente nel ciclo da noi costruito. La situazione che si è creata viene pertanto chiamata "ciclo infinito".
Il tasto RUN/STOP non ci permette di uscire dall'iterazione. Esiste una particolare routine in codice macchina facente parte dell'interprete BASIC che provvede a verificare l'eventuale pressione di quest'ultimo tasto, ma il nostro programma non contempla tuttavia la scansione della tastiera. Vi sono solamente due modi per uscire da una tale situazione. Il primo consiste nel premere simultaneamente i tasti RUN/STOP e RESET; questo arresterà l'esecuzione del programma restituendo il controllo a Tedmon. La seconda soluzione sarebbe rappresentata dallo spegnere il computer, ma si avrebbe come conseguenza la perdita irreversibile del programma memorizzato. Non esistono altri sistemi per uscire da una routine in linguaggio macchina prima che questa lo faccia automaticamente per mezzo di un RTS. Notate che a causa del JMP, il nostro programma non sarà mai in grado di effettuare da solo questa operazione, come nel caso del seguente listato BASIC:

```
 5 X=4
10 PRINT"HELLO";X
15 X=X+4
20 GOTO 10
30 END
```

Ovviamente, il comando END non verrà mai eseguito a causa del GOTO della linea 20. Per fare in modo che quest'ultimo programma stampi da HELLO 4 a HELLO 100, dobbiamo modificarlo come segue:

```
 5 X=4
10 PRINT"HELLO";X
15 X=X+4
20 IF X=104 GOTO 40
30 GOTO 10
40 END
```

In questo caso dalla linea 20 si salterà alla 40 soltanto se X=104, condizione per uscire dal programma attraverso il comando END. Finchè X sarà differente da 104, il ciclo continuerà ad essere ripetuto. Per tradurre il tutto in linguaggio macchina, abbiamo bisogno di una nuova istruzione che ci permetta di confrontare due valori (X e 104), e di un'altra che salti in funzione del risultato della comparazione (IF...GOTO 40).

Comparazione dei numeri

Abbiamo già avuto modo di trattare la nozione di flag. Si tratta di un valore binario a singolo bit immagazzinato all'interno del microprocesso-

re. Nel corso del capitolo 5 ci eravamo particolarmente occupati del flag di carry, il quale viene settato ad indicare un riporto durante l'effettuazione di addizioni multibyte (o di un prestito in sottrazioni multibyte). Il microprocessore dispone di sette differenti flags aventi ciascuno una determinata funzione, i quali sono tutti compresi all'interno di uno speciale registro denominato "registro di stato". Questi flags sono individualmente rappresentati dal loro bit, e dispongono di specifiche istruzioni per la loro gestione. Inoltre, il loro stato può essere influenzato più o meno direttamente dalla maggior parte delle altre istruzioni del linguaggio macchina (ne riparleremo più dettagliatamente nel capitolo 10). Ad esempio, ADC azzera o mette a uno il carry in funzione del risultato dell'addizione svolta. Similmente, l'istruzione CMP (CoMPare), la quale provvede a confrontare il contenuto dell'accumulatore con quello di un'altra locazione (dipendente dal modo di indirizzamento), manifesta i suoi risultati alterando i vari flags del registro di stato.

Istruzioni di salto condizionato

Riprendendo l'esempio precedente, la seconda nuova istruzione che ci necessita per scrivere il nostro programma deve saltare ad un determinato indirizzo di memoria in funzione del valore dei flags contenuti nel registro di stato. Questo tipo d'istruzione è denominata "salto condizionato" e differisce da JMP non soltanto a causa del condizionamento, ma perchè è la sola a fare uso dell'indirizzamento relativo. Tale indirizzamento è così chiamato perchè l'indirizzo di destinazione viene calcolato relativamente alla posizione dell'istruzione di salto condizionato. Parleremo più dettagliatamente dell'indirizzamento relativo e delle caratteristiche delle istruzioni di salto condizionato al termine di questo capitolo.

Il flag di zero

Per saltare ad un determinato indirizzo solamente nel caso in cui i due valori comparati per mezzo di CMP siano uguali, utilizzeremo l'istruzione BEQ (Branch on EQual).
Provate a digitare il seguente programma, che differisce dall'ultimo presentato per il fatto che la sua esecuzione viene sospesa quando il contenuto della locazione $0C00 diventa uguale a $80.

```
2000  LDA #$00
2002  STA $0C00
2005  LDA $0C00
2008  CMP #$80
200A  BEQ $2015
200C  CLC
```

```
200D  ADC #$04
200F  STA $0C00
2012  JMP $2005
2015  BRK
```

Abbiamo previsto di uscire dal ciclo quando si verifica una determinata condizione. Per rendere il tutto maggiormente efficiente, consideriamo ora il caso opposto, nel quale l'uscita dal ciclo avverrà solamente se la condizione non viene più verificata. La differenza è molto sottile, ma il seguente programma BASIC provvederà ad evidenziarla se confrontato con il precedente.

```
 5 X=4
10 PRINT"HELLO";X
15 X=X+4
20 IF X<>104 GOTO 10
30 END
```

L'equivalente in linguaggio macchina farà uso dell'istruzione di salto condizionato opposta a BEQ, ovvero BNE (Branch if Not Equal). Digitate quanto segue:

```
2000  LDA #$00
2002  STA $0C00
2005  LDA $0C00
2008  CLC
2009  ADC #$04
200B  STA $0C00
200E  LDA $0C00
2011  CMP #$80
2013  BNE $2005
2015  BRK
```

Come potete ben vedere, esistono molteplici modi per scrivere un programma. Quale sia giusto e quale sia sbagliato, nessuno può dirlo, ma la versione migliore è senza dubbio quella più facilmente leggibile e verificabile. Questo è decisamente il sistema più efficiente per costruire un codice che sia il più efficiente possibile.
Possiamo apprendere molto conoscendo la tecnica esecutiva di ogni istruzione. Ad esempio, CMP confronta due valori effettuando una sottrazione (accumulatore — memoria) senza tuttavia immagazzinare il risultato ottenuto; soltanto i flags del registro di stato vengono opportunamente alterati. Le istruzioni di salto condizionato precedentemente utilizzate (BEQ e BNE) non identificano la loro "uguaglianza" con i numeri direttamente comparati, ma con lo stato del flag di zero.

BEQ = Salta se il flag di zero è settato.
BNE = Salta se il flag di zero è azzerato.

Il flag di zero viene posto a uno nel caso i due valori comparati siano identici (accumulatore − memoria = 0), altrimenti viene azzerato. Lo stato di questo flag viene quindi verificato dalle istruzioni BEQ o BNE. Essendo CMP in effetti una sottrazione, anche il flag di carry viene interessato dal risultato. In altre parole, se la sottrazione effettuata da CMP necessita di un prestito (accumulatore minore della memoria), il carry viene azzerato, altrimenti (accumulatore maggiore o uguale alla memoria) viene posto a uno.
In conseguenza di quanto illustrato qui sopra, possiamo dunque affermare che l'istruzione CMP non testa soltanto l'uguaglianza di due valori, ma anche il loro ordinamento. Possiamo adesso scrivere il nostro nuovo programma BASIC:

```
 5 X=4
10 PRINT"HELLO";X
15 X=X+4
20 IF X<101 GOTO 10
30 END
```

In questo modo il suo funzionamento viene maggiormente evidenziato. Si vede infatti chiaramente come i valori di X superiori a 100 non vengano stampati. Per verificare che il contenuto dell'accumulatore sia minore di quello relativo alla locazione di memoria considerata, useremo CMP seguito da BCC (Branch on Carry Clear), in quanto il carry è azzerato in conseguenza di un avvenuto prestito. Per verificare invece che l'accumulatore sia maggiore o uguale alla memoria, ci avvaleremo di CMP seguito da BCS (Branch on Carry Set).

Indirizzamento relativo

Tutte le istruzioni di salto condizionato utilizzano un modo d'indirizzamento denominato "relativo" (ricordiamo che JMP non è un salto condizionato). Esso viene caratterizzato dal fatto che l'indirizzo di destinazione del salto viene calcolato a partire dalla posizione dell'istruzione generatrice. Tali particolari istruzioni sono tutte formate da due bytes: il primo per qualificare il tipo di salto ed il secondo per specificare indirettamente l'indirizzo. Questo viene determinato indicando il numero di locazioni comprese fra esso ed il primo byte successivo alla prima istruzione posta dopo il salto condizionato. Valori da $00 a $7F caratterizzano salti in avanti, mentre valori da $80 a $FF indicano invece un salto all'indietro di (256 − valore considerato) locazioni di memoria.

La peculiarità principale dei salti condizionati, in conseguenza del loro indirizzamento relativo, è rappresentata dalla totale indipendenza nei confronti del loro posizionamento in memoria. In caso di rilocamento del programma che li contiene, il linguaggio macchina rimane inalterato, mentre il disassemblato si adatta in accordo con la nuova posizione. Tale programma viene eseguito in modo costantemente analogo a prescindere dall'indirizzo di memoria a partire dal quale è stato memorizzato. Non altrettanto si può dire per quanto riguarda l'istruzione JMP, poichè quest'ultima impiega l'indirizzamento assoluto e pertanto non risulta direttamente rilocabile.

Riassunto del Capitolo 6

[1] L'istruzione JMP <indirizzo> equivale sostanzialmente al comando GOTO <numero di linea> del BASIC. Essa si occupa di far saltare l'esecuzione del programma all'indirizzo di memoria specificato.

[2] Per uscire da un ciclo infinito, premete insieme i tasti RUN/STOP e RESET.

[3] Il registro di stato del microprocessore dispone di sette flags (più uno non utilizzato), i quali vengono alterati da numerose istruzioni in codice macchina.

[4] Le istruzioni di salto condizionato si basano sullo stato del particolare flag al quale si riferiscono.
BEQ = Salta se il flag di zero è settato.
BNE = Salta se il flag di zero è azzerato.
BCS = Salta se il flag di carry è settato.
BCC = Salta se il flag di carry è azzerato.

[5] L'istruzione CMP provvede a confrontare due valori (eseguendo una sottrazione senza che il risultato venga immagazzinato). Unicamente i flags di zero e di carry vengono interessati.

	CARRY	ZERO
ACC minore di MEM	0	0
ACC uguale a MEM	?	1
ACC maggiore o uguale a MEM	1	?
ACC maggiore di MEM	1	0

[6] Il modo d'indirizzamento relativo, usato solamente dalle istruzioni di salto condizionato, specifica l'indirizzo di destinazione rispetto alla posizione dell'istruzione che lo impiega.

CAPITOLO 7

Contatori, iterazioni e puntatori

Contatori per il controllo di un ciclo

Supponiamo di voler moltiplicare due numeri fra loro. Non esiste nessuna singola istruzione in linguaggio macchina che provveda ad eseguire questa operazione, per cui dovremo scrivere un apposito programma. Potremmo, ad esempio, aggiungere un numero al totale, tante volte quante ne rappresenta il secondo numero.

```
10 A=7:B=3
20 T=T+A
30 T=T+A
40 T=T+A
50 PRINT"7*3=";T
```

Sarebbe molto più semplice e pratico (specialmente per grandi numeri) effettuare ciò per mezzo di un ciclo.

```
10 A=7:B=3
20 T=T+A
30 B=B-1
40 IF B<>0 GOTO 20
50 PRINT"7*3=";T
```

Desideriamo precisare che quello qui sopra riportato non vuole essere il miglior modo per effettuare una moltiplicazione, in quanto siamo interessati unicamente all'impiego di particolari istruzioni. Una tecnica notevolmente più efficace vi verrà presentata nel capitolo 10.

Utilizzo dell'accumulatore come contatore

In quest'ultimo breve programma, a differenza di tutti quelli visti precedentemente, sono presenti due variabili: 'A' che aggiungiamo di volta in volta al totale, e 'B' che controlla il ciclo. In questo caso, non è possibile uscire dal ciclo testando il totale, in quanto sarebbe necessario conoscere il risultato finale prima ancora di poter scrivere il programma. Prendendo spunto dagli esempi dello scorso capitolo, la nostra routine in linguaggio

macchina sarà dunque strutturata come segue:

```
      LDA #$00
      STA A
      LDA #$03
      STA B
ciclo LDA A
      CLC
      ADC #$07
      STA A
      LDA B
      SEC
      SBC #$01
      STA B
      BNE ciclo
      BRK
```

Utilizzo di bytes come contatori

La maggior parte di questo programma consiste nel trasferire valori dall'accumulatore alla memoria e viceversa. Dato che sovente si manifesta la necessità di aggiungere o sottrarre un'unità ad un valore utilizzato come contatore, sono state previste allo scopo speciali istruzioni. INC (INCrement memory) aumenta di uno il contenuto della specificata locazione di memoria, immagazzinando il risultato allo stesso indirizzo. Analogamente, DEC (DECrement memory) sottrae un'unità alla locazione considerata. Entrambe queste istruzioni non intervengono sul flag di carry, ma unicamente sul flag di zero. Digitate il seguente programma:

```
2000  LDA #$03
2002  STA $04
2004  LDA #$00
2006  CLC
2007  ADC #$07
2009  DEC $04
200B  BNE $2006
200D  STA $05
200F  BRK
```

$2000–$2004: inizializzazione.
$2006–$200B: ciclo chiuso fino all'annullamento di $05.
$200D–$200F: fine.

Utilizzando INC o DEC, possiamo impiegare come contatore qualsiasi locazione di memoria, mantenendo l'accumulatore disponibile per altre funzioni.

Esercizio

Riscrivete il precedente programma utilizzando INC e CMP per testare l'uscita dal ciclo.

I registri X e Y

Esistono sistemi estremamente più semplici per costruire dei contatori che non richiedono l'impiego di INC e DEC. Riferendoci al capitolo numero 2, avevamo affermato che il microprocessore 7501 dispone di tre registri ad uso generale, denominati A, X e Y. Sul registro A (accumulatore) ci siamo già sufficientemente soffermati; vediamo adesso come possono essere impiegati X e Y.
Quello che sappiamo di loro è che sono dei registri ad uso generale, contrariamente ad esempio al registro di stato (nonchè ad altri due che tratteremo in seguito). Quest'ultimo, in effetti, può venire usato soltanto per contenere i flags del microprocessore, e niente altro, mentre nell'accumulatore può essere immagazzinato un qualsiasi valore compreso fra 0 e 255 a scopo indeterminato. Analogamente, anche i registri X e Y possono contenere un qualsiasi numero di otto bits, tuttavia vi sono varie funzioni dell'accumulatore che essi non sono in grado di assolvere, come addizioni o sottrazioni. Il loro impiego principale, per il quale si rivelano estremamente utili, è appunto quello di contatori. Essi sono in grado di fare le seguenti operazioni (fra quelle precedentemente illustrate):

LDA	Carica l'accumulatore con un byte.
LDX	Carica X con un byte.
LDY	Carica Y con un byte.
STA	Scarica l'accumulatore in memoria.
STX	Scarica X in memoria.
STY	Scarica Y in memoria.
INC	Incrementa un byte.
INX	Incrementa X.
INY	Incrementa Y.
DEC	Decrementa un byte.
DEX	Decrementa X.
DEY	Decrementa Y.
CMP	Compara l'accumulatore con un byte.
CPX	Compara X con un byte.
CPY	Compara Y con un byte.

Le istruzioni d'incremento e decremento relative ad X e Y si intendono espresse in indirizzamento implicito.

Utilizzo del registro X come contatore

Riscriveremo adesso il nostro programma di moltiplicazione impiegando il registro X come contatore. Digitate:

```
2000  LDX #$03
2002  LDA #$00
2004  CLC
2005  ADC #$07
2007  DEX
2008  BNE $2004
200A  STA $03
200C  BRK
```

Questa nuova routine risulta leggermente più corta e sensibilmente più rapida dell'originale, ma per il resto si comporta esattamente nello stesso modo.
Provate adesso a riscrivere tutte quelle istruzioni che utilizzano il registro X, sostituendole per mezzo delle loro equivalenti relative al registro Y. Allenatevi anche a rimpiazzare l'accumulatore con X o Y ogni volta che risulta possibile nei programmi contenuti all'interno dei precedenti capitoli.

Trasferimento di blocchi di memoria

Come scrivereste un programma che sposti un blocco di memoria da una posizione ad un'altra, ad esempio da $2100—$2150 in $2200—$2250? Ovviamente non lo costruiremmo di certo nel modo seguente:

```
LDA  $2100
STA  $2200
LDA  $2101
STA  $2201
LDA  $2102
STA  $2202
```

ecc.

Tutto ciò sarebbe decisamente ridicolo, essenzialmente a causa della mole spropositata del programma che dovremmo scrivere. Si potrebbe tuttavia ridurlo a:

```
LDA  $2100
STA  $2000
```

seguito da determinate istruzioni che provochino un doppio incremento degli indirizzi di lettura e di destinazione ogni volta che una locazione viene trasferita. Questo è un concetto molto interessante da trattare: un programma che si automodifichi nel corso dell'esecuzione. A causa di questa sua caratteristica, il suo impiego è comunque estremamente pericoloso in quanto eventuali errori potrebbero comportare conseguenze altamente distruttive (riscrivere una zona inesatta del programma e quindi tentare di eseguirla potrebbe costringervi a spegnere e riaccendere il computer per poter proseguire). Tale tipo di codice risulta inoltre decisamente complicato da verificare. Benchè questo rappresenti un'affascinante tecnica di programmazione, vi sconsigliamo decisamente d'impiegarla in programmi seri. Esso non costituisce pertanto la soluzione al nostro problema.
Questa risiede in effetti nell'indirizzamento utilizzato. Originariamente, avevamo definito i modi d'indirizzamento come differenti sistemi e formati per accedere a dati o locazioni di memoria.

Indirizzamento implicito

Il dato viene specificato come parte integrante dell'istruzione. Esempi: SEC, DEY.

Indirizzamento relativo

L'indirizzo di destinazione viene determinato in funzione della posizione dell'istruzione. Utilizzato unicamente nei salti condizionati.

Indirizzamento assoluto

Il dato viene specificato attraverso il suo indirizzo di due bytes nel formato byte basso/byte alto.

Indirizzamento in pagina zero

Il dato viene specificato attraverso un indirizzo a singolo byte compreso nei primi 256 della memoria.

Indirizzamento indicizzato

La caratteristica di questo nuovo modo d'indirizzamento consiste nel determinare il dato utilizzato all'interno dell'istruzione addizionando all'indirizzo assoluto specificato un particolare byte di indice. Tale byte si trova immagazzinato in uno dei due registri X o Y (in funzione dell'istruzione impiegata).
Riprendendo la nostra precedente analogia con le caselle postali, per trovare quella corretta vi sono stati forniti due foglietti di carta: uno contenente un indirizzo di due bytes e l'altro un indice a singolo byte (0—255). L'indirizzo di destinazione si ricava addizionando insieme i due numeri; quello del secondo foglietto può tuttavia essere modificato nei successivi interventi (generalmente incrementato o decrementato di un'unità).

Utilizzo del registro X come indice

Grazie all'indirizzamento indicizzato, il nostro programma trasferitore di blocchi di memoria risulta notevolmente semplificato. Digitate quanto segue:

```
2000  LDX #$00
2002  LDA $2100,X
2005  STA $2200,X
2008  INX
2009  CPX #$51
200B  BNE $2002
200D  BRK
```

Notate che in questo caso, la forma mnemonica dell'indirizzamento indicizzato è costituita nell'ordine dal codice, dall'indirizzo assoluto, da una virgola separatrice e dal registro usato come indice. Le due istruzioni seguenti sono comunque equivalenti:

```
LDA $2100,X
LDA $2100,Y
```

A livello di linguaggio macchina, quella che cambia è l'istruzione vera e propria, non il campo dell'indirizzo. In questa particolare situazione i registri X e Y risultano pertanto intercambiabili, proprietà assolutamente non verificata nel caso generale.

Asimmetria dei comandi

Vi sarete forse domandati se sia possibile anche in quest'ultimo esempio rimpiazzare l'accumulatore con uno dei due registri X o Y. La risposta è

negativa. Non tutte le istruzioni possono infatti esprimersi in ognuno dei modi d'indirizzamento disponibili. L'utilizzo di un indice è peculiarità dell'accumulatore (LDA, STA), salvo particolari eccezioni molto limitative (è possibile impiegare "LDY indirizzo,X" ma non "STY indirizzo,X"). Una lista dettagliata di tutti gli indirizzamenti contemplati per ogni istruzione è contenuta nella prima appendice.

Ricerca all'interno della memoria

Sintetizzando le nozioni acquisite fino a questo punto, siamo adesso in grado di svolgere alcuni interessanti compiti con estrema facilità. Ad esempio, scriviamo un programma che ci permetta di localizzare il quarto riscontro del numero 9 all'interno della pagina di memoria (256 bytes) avente origine in $F000.
Innanzitutto determiniamo il primo riscontro di 9 a partire da $F00 digitando la seguente routine:

```
        LDY #$00
        LDA #$A9
ciclo   CMP $F000,Y
        BEQ trovato
        INY
        BNE ciclo
        BRK (non trovato)
trovato BRK (trovato un $A9 da $F000 a $F0FF)
```

Costruiamo ora una routine di conteggio per stabilire il momento corrispondente al quarto riscontro:

```
        LDX #$00
        ricerca
        INX
        CPX #$04
        BNE ricerca
```

Combinando insieme le due routines precedenti, otteniamo infine il seguente programma:

```
        LDX  #$00
        LDY  #$00
        LDA  #$A9
L40     CMP  $F000,Y
        BEQ  L100
L60     INY
        BNE  L40
```

```
        STX  $03
        RTS
L100    INX
        CPX  #$04
        BNE  L60
        STX  $03
        BRK
```

Al termine dell'esecuzione, il numero dei riscontri di $A9 nell'intervallo $F000—$F0FF viene immagazzinato nella locazione $03. Nel caso tale valore risulti esattamente uguale a quattro, l'indirizzo del quarto riscontro è $F000+Y. Qualora risulti invece minore di quattro, significa che $A9 è presente altrettante volte all'interno dell'intervallo in questione.
In base a questa configurazione, noi veniamo a conoscenza di quanti riscontri sono effettivamente avvenuti. Modificando la penultima istruzione STX $03 in STY $03, l'indirizzo del quarto riscontro ci verrà direttamente comunicato attraverso il contenuto del registro Y.

Utilizzo simultaneo di più indici

Scriveremo adesso un programma che utilizzerà simultaneamente entrambi i registri di indice per scopi differenti. Esso provvederà a costruire una lista di tutti i numeri inferiori a $38 compresi nell'intervallo $F000—$F0FF, e quindi a memorizzarla a partire da $3000.

```
        LDX  #$00
        LDY  #$FF
L30     INY
        LDA  $F000,Y
        CMP  #$38
        BCS  L90
        STA  $3000,X
        INX
L90     CPY  #$FF
        BNE  L30
        STX  $03
        BRK
```

Il registro X viene qui usato per puntare alla zona di memoria nella quale andare ad immagazzinare i nostri risultati. Notate che il registro Y viene inizializzato a $FF per essere poi incrementato e portato quindi a $00.
Per ricercare tutti i valori minori di $38 ci siamo avvalsi di CMP e BCS (riferirsi al capitolo 6) al fine di scavalcare le istruzioni d'immagazzinamento ed incremento del puntatore di controllo. Al termine dell'esecuzio-

ne, verificate attraverso Tedmon che a partire da $3000 siano effettivamente presenti solo numeri inferiori a $38.

Indirizzamento indicizzato in pagina zero

Tutte le istruzioni indicizzate impiegate in precedenza lo sono state a partire da un indirizzo di memoria assoluto (indirizzamento indicizzato assoluto). È tuttavia possibile indicizzare anche partendo da un indirizzo di pagina zero. Per riscrivere il programma appena illustrato in modo da fargli effettuare la sua ricerca all'interno dei primi 256 bytes di memoria, ci basterebbe modificare la quarta istruzione in LDA $00,Y. Ebbene, consultando l'elenco dei possibili indirizzamenti scopriamo che non è disponibile un "LDA pagina zero,Y" ma soltanto un "LDA pagina zero,X". Abbiamo in questo momento due possibilità di scelta. Da un punto di vista pratico, è probabilmente preferibile conservare l'istruzione d'indirizzamento indicizzato assoluto (LDA $0000,Y). Per ciò che riguarda invece le intenzioni più teoriche del nostro esercizio, opteremo per "LDA pagina zero,X" Digitate ed eseguite quanto segue:

```
LDY  #$00
LDX  #$FF
INX
LDA  $00,X
STA  $2200,Y
INY
CPY  #$FF
STY  $03
BRK
```

Questo vi dimostra quanto sia necessario prestare molta attenzione nella scelta dei registri. Benchè nella maggior parte dei casi essi siano intercambiabili, esistono tuttavia determinate istruzioni che non contemplano particolari modi d'indirizzamento con questo o quel registro. Vi consigliamo di riferirvi costantemente in sede di programmazione alla lista delle istruzioni presentata nell'appendice 1.

Riassunto del Capitolo 7

[1] INC aggiunge un'unità al contenuto della locazione di memoria specificata.

[2] DEC sottrae un'unità al contenuto della locazione di memoria specificata.

[3] Il flag di zero (ma non il carry) viene interessato da queste particolari istruzioni.

[4] Esse vengono impiegate soprattutto all'interno di cicli di conteggio in modo tale da mantenere disponibile l'accumulatore per altre funzioni.

[5] I registri X e Y del microprocessore possono essere utilizzati tanto come contatori quanto come indici.

[6] L'indirizzamento indicizzato addiziona il valore del registro di indice con l'indirizzo assoluto (o di pagina zero) specificato nell'istruzione, allo scopo di calcolare l'indirizzo finale di destinazione del dato trattato.

[7] La maggior parte delle istruzioni ricopre la medesima funzione se impiegata con l'accumulatore od i registri X e Y. Tuttavia, ne esiste qualcuna, insieme a particolari modi d'indirizzamento, che non risulta disponibile con tutti i registri. Mentre scrivete un programma, assicuratevi sempre che le istruzioni da voi utilizzate siano effettivamente implementate in quel particolare formato.

Utilizzo delle informazioni immagazzinate in tabelle

Uno dei più frequenti impieghi dei registri d'indice è costituito dall'accesso a tabelle d'informazione. Queste possono essere usate per varie ragioni: per contenere dati, indirizzi di subroutines, ed anche per facilitare difficoltose conversioni numeriche.

Visualizzazione grafica dei caratteri

Un esempio di conversione tramite tabella, per la quale non esistono formule di corrispondenza, consiste nel passaggio dal codice di schermo alla forma grafica del carattere da visualizzare. Normalmente di ciò se ne occupa direttamente il computer, per cui non dobbiamo preoccuparci della sua effettuazione. Tuttavia, quando viene inserito il modo grafico, il dispositivo di conversione viene disabilitato. Paragonando le locazioni di schermo ad altrettante caselle postali aventi lo sportello trasparente, possiamo identificare tale convertitore come una specie di filtro che provvede a trasformare il valore contenuto in ogni casella nella corrispondente rappresentazione grafica da visualizzare sullo schermo. In modo grafico, questo filtro rimane inattivo, e quello che vediamo sono i bits componenti i bytes immagazzinati nella memoria di schermo. Per ogni bit posto a uno viene acceso il corrispondente pixel, mentre quelli azzerati rappresentano un punto spento (avente il medesimo colore dello sfondo, e quindi invisibile).
A titolo di esempio, consideriamo il byte $11. La sua configurazione binaria è la seguente:

[0] [0] [0] [1] [0] [0] [0] [1]

La sua rappresentazione grafica è data nell'ordine da tre punti spenti, un punto acceso, altri tre punti spenti ed infine un ultimo punto acceso. In funzione della qualità dell'apparecchio televisivo da voi posseduto, sarete o meno in grado di distinguere i singoli punti formanti i caratteri presenti sullo schermo. Ogni carattere è composto da una matrice quadrata di 8*8 punti. Avendo appena determinato come un byte permetta di visualizzare otto punti (uno per bit), possiamo dedurre che la rappresentazione grafica di un carattere necessita di otto bytes posti uno sopra l'altro. Ad esempio,

la lettera A risulta strutturata nel modo seguente:

Matrice di punti 8*8 (Ø 1 2 3 4 5 6 7)	Configurazione binaria	Configurazione esadecimale
Ø	ØØØ11ØØØ	18
1	ØØ1ØØ1ØØ	24
2	Ø1ØØØØ1Ø	42
3	Ø111111Ø	7E
4	Ø1ØØØØ1Ø	42
5	Ø1ØØØØ1Ø	42
6	Ø1ØØØØ1Ø	42
7	ØØØØØØØØ	Ø

La sequenza degli otto bytes 18,24,42,7E,42,42,42,0 equivale esattamente a quanto contenuto nell'area di memoria addetta alla costruzione dei caratteri.

Memoria grafica

Come nel caso del convenzionale schermo di testo, lo schermo in alta risoluzione corrisponde ad una zona di memoria. L'informazione viene ad

	Colonna Ø	Colonna 1	Colonna 2	...	Colonna 39
Inizio della memoria grafica →	8192	82ØØ		...	
	8193	82Ø1		...	
	8194	82Ø2		...	
	8195	82Ø3		...	
RIGA 0	8196	8204		...	
	8197	82Ø5		...	
	8198	8206		...	
	8199	82Ø7		...	
↓					
				...	16184
				...	16185
				...	16186
				...	16187
RIGA 24				...	16188
				...	16189
				...	16190
				...	16191 ← Fine della memoria grafica

esso indirizzata memorizzando opportunamente valori numerici in tale zona. Lo schermo grafico ha origine a partire dalla locazione 8192 e si estende lungo un'area di 8000 bytes (64000 punti). È strutturato su quaranta colonne per venticinque linee di caratteri, ogni carattere essendo formato da otto righe di otto punti ciascuna. Ogni punto dello schermo ha inoltre la facoltà di essere autonomamente spento o acceso in funzione delle esigenze di programmazione.

Indirizzamento indiretto indicizzato

Talvolta può accadere di non essere in grado di determinare a quale tabella accedere per prelevare i dati richiesti. In altre parole, consideriamo un programma che vi permetta di decidere se stampare un messaggio in maiuscolo oppure in minuscolo. In base alla scelta effettuata, esso accederà ad una delle due tabelle contenenti i caratteri da visualizzare. Questa operazione potrebbe essere compiuta per mezzo di due programmi simili fra loro, ciascuno dei quali accede ad una specifica tabella; la determinazione di quello da eseguire avviene inizialmente in funzione della scelta dell'utente. Ovviamente questa tecnica risulterebbe caratterizzata da un notevole spreco di memoria, pertanto non avrebbe alcuna utilità pratica. Fortunatamente, è stato previsto allo scopo uno speciale modo d'indirizzamento denominato "indiretto indicizzato". Esso risulta molto simile all'indirizzamento indicizzato assoluto, differenziandosi da quest'ultimo per il fatto che l'indirizzo assoluto non è direttamente definito all'interno dell'istruzione, ma viene immagazzinato in due successive locazioni di pagina zero. Queste puntano indirettamente ad un indirizzo di memoria immagazzinato nel formato byte basso/byte alto, al quale viene successivamente addizionato il contenuto del registro Y, utilizzato come indice, in modo da ricavare l'indirizzo finale al quale si riferisce l'istruzione (l'indirizzamento indiretto indicizzato utilizza sempre il registro Y come indice). La forma mnemonica di questo modo d'indirizzamento è:

QQQ (PZ),Y

dove QQQ rappresenta l'istruzione utilizzata e PZ una locazione di pagina zero. Il registro Y viene posto all'esterno delle parentesi per segnalare che la puntata indiretta ha la priorità sull'aggiunta dell'indice. Provate a digitare il seguente programma:

```
LDA  #$00
STA  $03
LDA  #$30
STA  $04
LDY  #$00
```

```
ciclo LDA  ($03),Y
      STA  $2600,Y
      INY
      CPY  #$FF
      BNE  ciclo
      BRK
```

In questo esempio, il riferimento indiretto punta a $3000 attraverso le locazioni di pagina zero $03 e $04.

Istruzioni di trasferimento dei registri

Esistono speciali istruzioni che permettono il trasferimento d'informazione dall'accumulatore ai registri X o Y, e viceversa. Le quattro disponibili sono:

TAX (Trasferisce l'accumulatore nel registro X)
TAY (Trasferisce l'accumulatore nel registro Y)
TXA (Trasferisce il registro X nell'accumulatore)
TYA (Trasferisce il registro Y nell'accumulatore)

Queste istruzioni vengono principalmente utilizzate ogni volta che un contatore od un indice richiedono particolari manipolazioni aritmetiche da effettuarsi esclusivamente all'interno dell'accumulatore e quindi ritornare all'origine. Notate che non è stata prevista nessuna istruzione di trasferimento diretto fra i registri X e Y; qualora si desideri eseguire una tale operazione, è necessario passare attraverso l'accumulatore.
Vi sono due ulteriori modi d'indirizzamento non ancora trattati che andremo ad illustrarvi brevemente. Il primo di essi è denominato "indicizzato indiretto", da non confondersi con quello "indiretto indicizzato" appena visto. L'ordine dei termini di descrizione caratterizza l'ordine di esecuzione delle operazioni. La forma mnemonica di quest'ultimo indirizzamento è la seguente:

QQQ (PZ,X)

dove QQQ rappresenta l'istruzione e PZ una locazione di pagina zero. L'indirizzo al quale si riferisce viene indirettamente puntato dalle locazioni (PZ+X) e (PZ+X+1) contenenti rispettivamente il suo byte basso ed il suo byte alto. In questo caso l'operazione d'indicizzazione è prioritaria. Immaginate adesso di disporre di una tabella d'indirizzi contenuta in pagina zero. Questi indirizzi puntano ciascuno ad un insieme di dati o ad una specifica tabella. Per determinare la locazione iniziale di queste ultime, assegnate al registro X il corretto valore da sommare a PZ in modo da utilizzare quel particolare indirizzo della tabella di pagina zero che punti effettivamente a tale locazione.

Indirizzamento indiretto

L'ultimo indirizzamento che tratteremo è denominato semplicemente "indiretto". Il suo impiego è limitato esclusivamente all'istruzione JMP. Simile al comando GOTO del BASIC in indirizzamento assoluto, questa istruzione fa riferimento in indirizzamento indiretto ad una locazione che, unitamente alla successiva, contiene l'indirizzo effettivo di destinazione del salto immagazzinato nel consueto formato byte basso/byte alto. Vediamo un esempio:

```
LDA  #$00
STA  $2000
LDA  #$30
STA  $2001
JMP  ($2000)
```

Le locazioni $2000 e $2001 contengono rispettivamente i valori $00 e $30, byte basso e byte alto dell'indirizzo assoluto $3000, il quale rappresenta la destinazione effettiva dell'istruzione di salto indiretto JMP ($2000).
Questo modo d'indirizzamento viene generalmente impiegato nelle cosiddette entrate od uscite "vettorizzate". Supponiamo di voler stampare un messaggio tanto sul video quanto sulla stampante. Mentre nel primo caso possono essere sfruttate apposite routines della memoria ROM, nel secondo è necessario eseguire un programma esterno. Utilizzando un'uscita vettorizzata, non sarà necessario predisporre una specifica istruzione di chiamata per ciascuna routine, ma si potrà utilizzare sempre il medesimo salto indiretto a condizione di sostituire l'indirizzo di destinazione nelle relative locazioni puntatrici in funzione della routine prescelta.
Eccovi in conclusione un elenco dei differenti modi d'indirizzamento contemplati dal microprocessore 7501.

IMPLICITO	QQQ	
ASSOLUTO	QQQ	indirizzo
IN PAGINA ZERO	QQQ	PZ
IMMEDIATO	QQQ	# byte
RELATIVO	BQQ	byte
INDICIZZATO ASSOLUTO,X	QQQ	indirizzo,X
INDICIZZATO ASSOLUTO,Y	QQQ	indirizzo,Y
INDICIZZATO IN PAGINA ZERO,X	QQQ	PZ,X
INDICIZZATO IN PAGINA ZERO,Y	QQQ	PZ,Y
INDIRETTO INDICIZZATO	QQQ	(PZ),Y
INDICIZZATO INDIRETTO	QQQ	(PZ,X)
INDIRETTO	JMP	(indirizzo)
ACCUMULATORE (vedere capitolo 10)	QQQ	A

Riassunto del Capitolo 8

[1] In modo grafico è possibile "vedere" il contenuto della memoria di schermo. Un bit rappresenta un pixel (punto dello schermo).

[2] I caratteri sono definiti all'interno di una matrice quadrata di 8*8 punti.

[3] La memoria di schermo in modo grafico si articola carattere per carattere dall'alto in basso e da sinistra verso destra.

[4] I due sets di caratteri sono immagazzinati in ROM.

[5] I registri di indice vengono impiegati per accedere alle tabelle (fra le altre cose) attraverso vari modi d'indirizzamento indicizzati.

[6] Nel normale indirizzamento indicizzato, il contenuto del registro di indice viene addizionato ad un indirizzo assoluto (o di pagina zero) allo scopo di calcolare la locazione di destinazione.

[7] Nell'indirizzamento indiretto indicizzato, la locazione di destinazione viene calcolata sommando il contenuto del registro Y all'indirizzo assoluto immagazzinato nelle due locazioni di pagina zero puntate dall'istruzione.

[8] Nell'indirizzamento indicizzato indiretto, la locazione di destinazione viene calcolata sommando il contenuto del registro X all'indirizzo di pagina zero specificato nell'istruzione. La locazione ottenuta, unitamente alla successiva, contiene l'indirizzo di destinazione nel formato byte basso/byte alto.

[9] Il computer non è in grado di segnalare differenze fra dati significativi e dati senza senso.

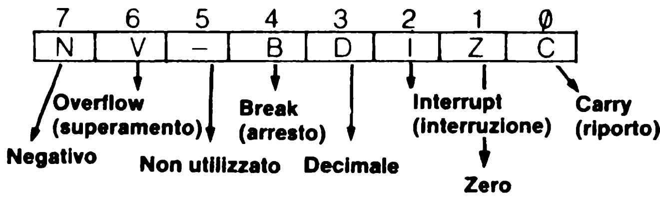
[10] Le istruzioni TAX, TAY, TXA e TYA vengono utilizzate per trasferire informazione dall'accumulatore ai registri di indice e viceversa.

[11] L'indirizzamento indiretto (solo per JMP) utilizza il contenuto di due bytes consecutivi, immagazzinati in una qualsiasi zona della memoria, per ricavare l'indirizzo di destinazione del salto.

CAPITOLO 9

Codici di stato del microprocessore

Abbiamo menzionato nei capitoli 5 e 6 il concetto di "flag" del microprocessore. Abbiamo parlato dei flags di carry e di zero e trattato dei vari salti condizionati, nonchè di tutte le altre istruzioni che ne influenzano lo stato (SEC, CLC, BCS, BCC, BEQ, BCC). Abbiamo infine precisato che questi due flags, unitamente ad altri cinque, sono immagazzinati in uno speciale registro del microprocessore denominato "registro di stato". Quest'ultimo, come qualsiasi altro registro o locazione di memoria, è formato da otto bits ognuno dei quali rappresenta un particolare flag.



Nell'appendice [1] troverete un elenco dettagliato di ogni singola istruzione comprendente i flags interessati per ciascuna di esse.

Il flag di Carry

Viene settato od azzerato per indicare una condizione di riporto od un prestito dell'ottavo bit di un byte dentro il "nono" bit. Dato che quest'ultimo non esiste realmente, è stato previsto il flag di carry allo scopo di permettere l'eventuale inclusione di tale particolare condizione in calcoli futuri. Il flag di carry può inoltre essere direttamente settato od azzerato rispettivamente per mezzo delle istruzioni SEC e CLC. Un programma ha infine la facoltà di verificare lo stato di tale flag attraverso le istruzioni di salto condizionato BCS (carry settato) o BCC (carry azzerato).

Il flag di zero

Viene settato od azzerato in funzione del risultato di un'operazione, di una comparazione o di un trasferimento d'informazione. Un programma

ha la facoltà di verificare lo stato di tale flag attraverso le istruzioni di salto condizionato BEQ (zero settato) o BNE (zero azzerato).

Assegnamento del flag di break

Settando il flag di break per mezzo dell'istruzione BRK viene provocata l'effettuazione di un cosiddetto "interrupt" (interruzione). Questo concetto sarà ulteriormente approfondito nel capitolo 11. L'istruzione BRK causa l'arresto immediato dell'esecuzione del vostro programma in linguaggio macchina ed il susseguente salto indiretto all'indirizzo contenuto nelle locazioni $FFFE e $FFFF. Queste locazioni della memoria ROM puntano ad una routine che provoca il ritorno al BASIC. L'impiego dell'istruzione BRK rappresenta un'ottima tecnica di verifica dei programmi; inserendola in punti specifici, è possibile localizzare ed isolare molto più facilmente eventuali errori. Essa vi permette di arrestare l'esecuzione di un programma e verificare che in quel momento il contenuto delle sue variabili corrisponda a quello previsto.

Il flag di interrupt

Può essere settato od azzerato rispettivamente per mezzo delle istruzioni SEI e CLI. Quando viene settato, tale flag provvede a disabilitare alcuni particolari tipi d'interruzione (vedere il capitolo 11).

Il flag decimale

Può essere settato od azzerato rispettivamente per mezzo delle istruzioni SED e CLD. Quando viene settato, il microprocessore entra nel modo di numerazione decimale, conosciuto anche come modo BCD (Binario Codificato in Decimale). Questo particolare modo costituisce un metodo di rappresentazione dei numeri decimali all'interno della memoria del computer. In esso, le cifre esadecimali da 0 a 9 vengono lette come le loro equivalenti decimali, mentre quelle da A a F non hanno alcun significato. Per meglio illustrarvi questo concetto, vi presentiamo ora una piccola tabella di corrispondenza:

BINARIO	ESADECIMALE	DECIMALE BCD
00000000	00	0
00000001	01	1
00000010	02	2
00000011	03	3
00000100	04	4
00000101	05	5
00000110	06	6

00000111	07	7
00001000	08	8
00001001	09	9
00010000	10	10
00010001	11	11
00100010	22	22
01000011	43	43
10011000	98	98
10011001	99	99

Come potete ben vedere, sono andati persi sei possibili valori compresi fra \$09 e \$10. In modo decimale, il microprocessore diventa in grado di effettuare direttamente addizioni e sottrazioni con numeri BCD.

Flag decimale = 0	Flag decimale = 1
17	17
+26	+26
3D	43

Le caratteristiche negative del modo decimale BCD consistono in un sensibile spreco di memoria ed in un'estrema lentezza operativa ad alti livelli (addizioni e sottrazioni semplici escluse). Generalmente è più pratico lavorare in esadecimale convertendo i risultati al momento dell'uscita, motivo per cui il modo decimale viene raramente utilizzato. Esercitatevi a trasformare alcuni dei programmi presentati nel capitolo 5 e confrontate i risultati ottenuti.

Il flag negativo

Sappiamo benissimo che il massimo valore immagazzinabile in una singola locazione di memoria è 255. Avevamo inoltre segnalato come fosse possibile rappresentare numeri più grandi per mezzo di due bytes contigui, ma non avevamo detto nulla a proposito dei valori minori di zero. Senza rendercene conto, sono stati brevemente utilizzati nel corso del sesto capitolo. Numeri da 0 a 255 possono rappresentare istruzioni, caratteri, indirizzi, valori matematici, ma il comportamento del microprocessore è sempre lo stesso: esso infatti esegue ciecamente tutte le manipolazioni che gli ordiniamo, senza distinzioni. Riprendendo la definizione dei salti condizionati, l'indirizzamento relativo utilizza i valori da \$00 a \$7F per quelli in avanti (positivi) ed i valori da \$80 a \$FF per quelli all'indietro (negativi). Questo sistema di numerazione è puramente arbitrario, ma dato che funziona è matematicamente valido per rappresentare numeri positivi e negativi. Tale sistema è denominato "complemento a due". Per definizione, il complemento a due di un numero binario si ottiene inver-

tendo i suoi bits ed aggiungendo quindi un'unità. Ricordatevi di consultare l'apposita tabella dell'appendice [4] per calcolare il complemento a due di un numero. Vediamo un esempio:

```
3 = 00000011  -->   11111100
                          +1
                  ----------
                  = 11111101 = $FD = (-3)
```

Utilizzando questa rappresentazione, è possibile constatare come ogni byte contenente un valore maggiore di 127 (bit più significativo settato) caratterizzi un numero negativo, mentre ogni byte contenente un valore minore di 128 (bit più significativo azzerato) caratterizzi invece un numero positivo.

```
1 x x x x x x x (NEGATIVO)
0 x x x x x x x (POSITIVO)
```

Il flag negativo del registro di stato viene automaticamente settato ogniqualvolta il risultato di un'operazione, una comparazione od un trasferimento è appunto negativo. Dato che il microprocessore non è in grado di determinare se il valore trattato rappresenti un carattere, un'istruzione o altro, esso provvederà a settare il flag negativo ogni volta che il bit più significativo (bit sette) del byte in uso è anch'esso settato. In altre parole, il flag negativo rappresenta sempre una copia del bit più significativo del risultato di un'operazione.

Visto che il bit più significativo è diventato un bit di segno, ce ne rimangono soltanto sette nei quali immagazzinare il nostro numero. Con sette bits è possibile rappresentare valori compresi fra 0 e 127, ma dato che 0 = −0, ne aggiungiamo uno in più dal lato negativo. Pertanto, la numerazione in complemento a due è in grado di rappresentare qualunque numero da −128 a +127 per mezzo di un singolo byte. Vediamo adesso alcuni esempi di calcolo utilizzanti il nostro nuovo sistema di numerazione in complemento a due.

[1] Positivo + positivo (risultato minore di 127)

```
  00000111          (+ 7)
+ 00001001        + (+ 9)
----------        -------
= 00010000        = (+16)
```

Carry = 0
Overflow = 0
Negativo = 0

[2] Positivo + negativo (risultato negativo)

00000111	(+ 7)
+ 11110100	+ (−12)
= 11111011	= (− 5)

Carry = 0
Overflow = 0
Negativo = 1

[3] Positivo + negativo (risultato positivo)

00000111	(+ 7)
+ 11111101	+ (− 3)
= 00000100 (carry = 1)	= (+ 4)

Carry = 1
Overflow = 0
Negativo = 0

[4] Positivo + positivo (risultato maggiore di 127)

01110011	(+115)
+ 00110001	+ (+ 49)
= 10100100	= (− 92)

Carry = 0
Overflow = 1
Negativo = 1

Attenzione: l'ultimo risultato ottenuto è SBAGLIATO.

Tutto sembra svolgersi correttamente, tranne che in quest'ultimo esempio. Avevamo precedentemente affermato che il complemento a due può gestire unicamente numeri compresi fra −128 e +127. Il risultato della nostra operazione avrebbe dovuto essere 164. Così come per contenere valori superiori a 255 sono necessari due bytes, oltrepassando in questo caso il limite di +127 ricadiamo nel medesimo problema. In una normale configurazione binaria, un riporto dal bit sette viene effettuato attraverso il carry. Nel complemento a due abbiamo soltanto sette bits più uno di segno, per cui quello più significativo risulta essere il bit sei. Il microprocessore, non sapendo che stiamo utilizzando questo particolare sistema di numerazione, riporta come di consueto il bit sei nel bit sette. Essendo il

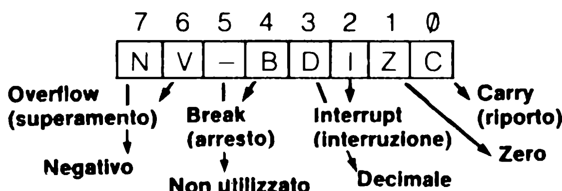
bit sette riservato al segno, il risultato viene influenzato; esso provvede allora a settare il flag di overflow per segnalare quest'anomala accensione del bit sette.

Il flag di Overflow

Questo flag viene settato ad indicare un riporto dal bit sei al bit sette. La funzione principale di tale flag consiste nel segnalare l'accidentale cambiamento di segno provocato da un superamento (overflow). Il corretto risultato dell'operazione si ottiene quindi invertendo il bit di segno ed aggiungendo il riporto se necessario. Il valore −92 (10100100) ottenuto attraverso la nostra precedente operazione diventa pertanto 1*128 (riporto) + 36 (00100100) = 164, corrispondente all'esatto totale della somma 115 + 49.
Un programma può testare lo stato del flag negativo attraverso le istruzioni di salto condizionato BMI (flag settato) e BPL (flag azzerato). Analogamente, è possibile testare lo stato del flag di overflow per mezzo di BVS (flag settato) e BVC (flag azzerato). Il flag di overflow può infine venire direttamente azzerato utilizzando l'istruzione CLV (CLear oVerflow).

Riassunto del Capitolo 9

[1] Il microprocessore contiene uno speciale registro di controllo denominato "registro di stato".



[2] Carry.
SEC = setta il flag di carry.
CLC = azzera il flag di carry.
BCS = salta se il flag di carry è settato.
BCC = salta se il flag di carry è azzerato.

[3] Zero.
BEQ = salta se il flag di zero è settato.
BNE = salta se il flag di zero è azzerato.

[4] L'istruzione BRK provvede a settare il flag di break arrestando immediatamente il microprocessore. Viene soprattutto utilizzata per verificare un programma.

[5] Interrupt (vedere capitoli 11 e 12).
SEI = setta il flag disabilitatore dell'interrupt.
CLI = azzera il flag disabilitatore dell'interrupt.

[6] Decimale.
SED = abilita il modo decimale BCD.
CLD = disabilita il modo decimale BCD.

[7] La numerazione in complemento a due permette di rappresentare valori compresi fra −128 e +127.

[8] Negativo.
BMI = salta se il flag negativo è settato.
BPL = salta se il flag negativo è azzerato.

[9] Overflow.
CLV = azzera il flag di overflow.
BVS = salta se il flag di overflow è settato.
BVC = salta se il flag di overflow è azzerato.

Operatori logici e manipolatori di bits

Alterazioni di bits in memoria

In questo capitolo vedremo un gruppo d'istruzioni, sensibilmente differenti da quelle trattate in precedenza, che risultano assolutamente fondamentali per le attività del computer. Stiamo parlando dei cosiddetti "operatori logici" o "booleani". Si tratta delle istruzioni AND (AND logico), ORA (OR logico) e EOR (OR esclusivo logico). Le loro funzioni vengono costruite per mezzo di una circuitazione estremamente semplice, e praticamente la maggior parte di tutte le altre si ottiene combinando in serie tali circuiti. Noi ci limiteremo comunque ad analizzare questi operatori logici esclusivamente dal punto di vista della programmazione, senza preoccuparci della loro struttura elettronica. Sappiamo bene che una locazione di memoria è composta da otto bits contigui.



Per meglio illustrare le funzioni di questi operatori logici, restringeremo la descrizione ad un singolo bit, considerando che in realtà tali funzioni fanno riferimento a tutti e otto i bits contemporaneamente. Un operatore logico conferma il concetto di operazione in quanto a partire da due dati ne ricava un terzo come risultato. Questi dati, in quanto cifre binarie, possono essere unicamente zero o uno. Per definire una funzione logica ci avvaleremo di una tabella di verità che provvede ad illustrare tutte le possibili combinazioni d'ingresso ed i loro corrispondenti risultati d'uscita.

INGRESSO 1 / INGRESSO 2	Ø	1
Ø	USCITA PER Ø, Ø	USCITA PER Ø, 1
1	USCITA PER 1, Ø	USCITA PER 1, 1

And logico

La prima istruzione che tratteremo è AND. Essa effettua un'operazione di AND logico dell'accumulatore con la specificata locazione di memoria o valore, riponendovi quindi il risultato ottenuto. Questo è uguale a uno soltanto nel caso entrambi i bits in ingresso siano anch'essi uguali a uno. Eccovi la sua tabella di verità:

ACCUMULATORE \ MEMORIA	Ø	1
Ø	Ø	Ø
1	Ø	1

Estendendo l'operazione a tutti e otto i bits, otteniamo un esempio di questo tipo:

	Ø	1	1	Ø	1	Ø	1	1
AND	1	Ø	1	1	1	Ø	1	Ø
=	Ø	Ø	1	Ø	1	Ø	1	Ø

Il flag di zero viene settato se il risultato dell'operazione è nullo. Questo accade solo nel caso non siano presenti nei due bytes in ingresso coppie di bits corrispondenti entrambi posti a uno.
L'istruzione AND è molto utile per creare una maschera che spenga particolari bits all'interno di un byte senza influire sullo stato degli altri. Supponiamo ad esempio di voler azzerare il bit tre, cinque e sei di una qualsiasi locazione di memoria. Costruiremo quindi una maschera avente soltanto questi tre bits posti a zero, ed effettueremo un AND logico di essa con il byte in questione.

	7	6	5	4	3	2	1	Ø	
MASCHERA =	1	Ø	Ø	1	Ø	1	1	1	= $97

AND #$97.

L'istruzione AND #$97 provvede ad azzerare il bit tre, cinque e sei di un qualunque valore contenuto nell'accumulatore.

Or logico

La seconda istruzione che andremo ad illustrarvi è ORA. Essa effettua un'operazione di OR logico dell'accumulatore con la specificata locazione di memoria o valore, riponendovi quindi il risultato ottenuto. Questo è uguale a zero soltanto nel caso entrambi i bits in ingresso siano anch'essi uguali a zero. Eccovi la sua tabella di verità:

OR

ACCUMULATORE \ MEMORIA	0	1
0	0	1
1	1	1

Estendendo l'operazione a tutti e otto i bits, otteniamo un esempio di questo tipo:

	0	1	0	1	0	0	1	0
ORA	0	0	1	1	1	0	1	0
=	0	1	1	1	1	0	1	0

Il flag di zero viene settato se entrambi i bytes in ingresso sono nulli, ovvero con tutti i bits azzerati.
L'istruzione ORA è molto utile per creare una maschera che accenda particolari bits all'interno di un byte senza influire sullo stato degli altri.
Supponiamo ad esempio di voler settare il bit due, tre e sette di una qualsiasi locazione di memoria. Costruiremo quindi una maschera avente soltanto questi tre bits posti a uno, ed effettueremo un OR logico di essa con il byte in questione.

	7	6	5	4	3	2	1	0		
MASCHERA =	1	0	0	0	1	1	0	0	=	$8C

ORA #$8C.

L'istruzione ORA #$8C provvede a settare il bit due, tre e sette di un qualunque valore contenuto nell'accumulatore.

Or esclusivo logico

L'ultimo degli operatori logici che ancora ci resta da vedere è EOR. Esso effettua un'operazione di OR esclusivo logico dell'accumulatore con la specificata locazione di memoria o valore, riponendovi quindi il risultato ottenuto. Questo è uguale a zero soltanto nel caso entrambi i bits in ingresso siano uguali fra loro. Eccovi la sua tabella di verità:

EOR

ACCUMULATORE \ MEMORIA	Ø	1
Ø	Ø	1
1	1	Ø

Estendendo l'operazione a tutti e otto i bits, otteniamo un esempio di questo tipo:

	1	Ø	1	1	1	Ø	Ø	1
EOR	1	Ø	1	Ø	Ø	1	Ø	1
=	Ø	Ø	Ø	1	1	1	Ø	Ø

La tecnica di mascheramento applicata all'istruzione EOR permette d'invertire particolari bits all'interno di un byte senza influire sullo stato degli altri. Supponiamo ad esempio di voler invertire il bit uno, due, e quattro di una qualsiasi locazione di memoria. Costruiremo quindi una maschera avente soltanto questi tre bits posti a uno, ed effettueremo un EOR logico di essa con il byte in questione.

	7	6	5	4	3	2	1	Ø	
MASCHERA =	Ø	Ø	Ø	1	Ø	1	1	Ø	= $16

EOR #$16.

L'istruzione EOR #$16 provvede ad invertire il bit uno, due e quattro di un qualunque valore contenuto nell'accumulatore.

Per verificare il funzionamento dei tre operatori logici, provate a digitare

il seguente programma:

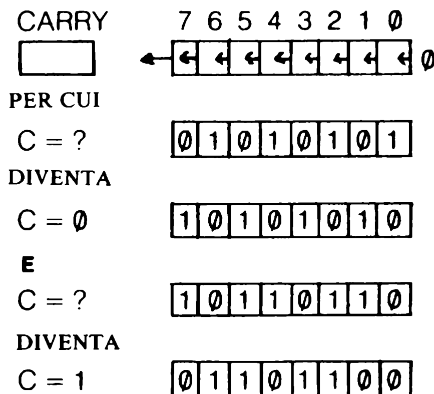
```
2000  LDA #$CA    (A = $CA)    (11001010)
2002  AND #$9F    (A = $8A)    (10001010)
2004  STA $03     (A in $03)   (10001010)
2006  LDA #$2     (A = $A2)    (10100010)
2008  ORA #$84    (A = $A6)    (10100110)
200A  EOR $03     (A = $2C)    (00101100)
200C  STA $03     (A in $03)   (00101100)
200E  BRK
```

L'istruzione bit

Utilissima istruzione del microprocessore 7501, BIT effettua un'operazione di AND logico dell'accumulatore con una determinata locazione di memoria senza tuttavia reimmagazzinarvi il risultato, ma limitandosi a settare eventualmente il flag di zero nel caso questo sia nullo. Essa provvede inoltre a copiare il bit sette nel flag negativo ed il bit sei nel flag di overflow.

Rotazione dei bits all'interno di un byte

Tratteremo adesso altre quattro istruzioni manipolatrici di bits, ed alcune loro conseguenze. La prima che andremo ad illustrarvi è ASL (Arithmetic Shift Left). Essa provvede a spostare tutti i bits all'interno di un byte di un posto verso sinistra, introducendo uno zero in quello meno significativo ed immagazzinando quello più significativo nel flag di carry.



Riprendendo la struttura di un byte, ad ogni suo bit viene associata una potenza di due in funzione del rango occupato (da 0 a 7). Avrete notato come il valore di ciascuno di essi risulti doppio di quello immediatamente alla sua destra. Da ciò ne ricaviamo ad esempio che:

```
00000001 * 2 = 00000010
00001000 * 2 = 00010000
00111001 * 2 = 01110010
```

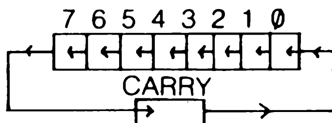
L'operazione necessaria a moltiplicare ogni byte per due corrisponde esattamente a quella eseguita da ASL.
Digitate il seguente programma:

```
2000 LDA #$0A
2002 ASL
2003 STA $03
2005 BRK
```

Al termine dell'esecuzione, la locazione $03 dovrebbe contenere il valore $14 ($0A * 2 = $14). Notate che l'indirizzamento utilizzato per ASL non è quello implicito, in quanto tale istruzione può riferirsi anche ad una qualsiasi locazione di memoria. È possibile infine moltiplicare un valore per una qualsiasi potenza di due mettendo in sequenza tante istruzioni ASL quante ne rappresenta l'esponente. Ad esempio, la moltiplicazione per otto viene eseguita attraverso tre ASL consecutive.

Rotazione con carry

Come nel caso dell'addizione, potremmo avere necessità di moltiplicare numeri maggiori di 255. Per effettuare ciò esiste una speciale istruzione di rotazione che utilizza il flag di carry tanto come estremità di entrata quanto come estremità di uscita.



Questa istruzione è ROL (Rotate One bit Left). Il seguente programma provvede a moltiplicare il numero di due bytes $170A per quattro:

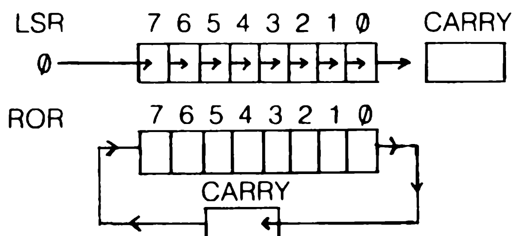
```
2000 LDA #$17
2002 STA $04
```

```
2004 LDA #$0A
2006 ASL
2007 ROL $04
2009 ASL
200A ROL $04
200C STA $03
200E BRK
```

Per evitare sovrapposizioni nell'accumulatore, abbiamo utilizzato ROL con l'indirizzamento in pagina zero, il quale provvede ad effettuare la rotazione direttamente all'interno della locazione specificata. Abbiamo inoltre ruotato entrambi i bytes del numero da moltiplicare una prima volta, e successivamente una seconda. Ruotare due volte il byte basso e quindi due volte il byte alto non avrebbe funzionato, in quanto il bit più significativo del byte basso sarebbe andato perso al momento dell'utilizzo del carry da parte della seconda ASL.

Rotazione verso destra

LSR e ROR sono le istruzioni rispettivamente equivalenti a ASL e ROL, differenziandosi da queste ultime per lo spostamento dei bits verso destra.



Esattamente come le due precedenti istruzioni venivano caratterizzate sotto forma di moltiplicazione per due, queste vengono similmente associate alla divisione per due, risultando altrettanto estendibili nei confronti di numeri composti da più bytes. Al termine della divisione, il risultato immagazzinato nella locazione utilizzata o nell'accumulatore rappresenta la parte intera del quoziente reale, mentre il numero dei bits spostati corrisponde al resto. Vediamo un esempio di divisione semplice per mezzo di LSR: $1D / $08 = 3 (resto = 5)

$1D = 29 = 00011101 / 2 = 00001110 = 14 (resto 1 = 1)
/ 4 = 00000111 = 7 (resto 01 = 1)
/ 8 = 00000011 = 3 (resto 101 = 5)

Sebbene le istruzioni di spostamento (ASL, LSR) e rotazione (ROL, ROR) vengano utilizzate nella maggior parte dei casi a scopo aritmetico, non dimenticatevi delle altre loro applicazioni, come ad esempio far slittare un bit di estremità nel carry per poterlo quindi testare attraverso un salto condizionato.

Moltiplicazione intelligente

Abbiamo precedentemente affermato come, spostando più volte i bits verso sinistra, sia possibile moltiplicare numeri per una qualsiasi potenza di due. Tali potenze rappresentano direttamente ogni singolo bit all'interno di un byte (1,2,4,...128). Nel terzo capitolo avevamo inoltre dimostrato come, addizionando fra loro queste potenze di due, fosse possibile rappresentare qualsiasi numero compreso fra 0 e 255. Se adesso moltiplichiamo per ciascuno di questi valori e sommiamo i risultati, equivale a moltiplicare direttamente per ogni numero da 0 a 255. Vediamo un esempio:

```
$16 * $59 = 00010110 * $59
          + 00000100 * $59
          + 00000010 * $59
          = (16 * $59) + (4 * $59) + (2 * $59)
```

Ricadendo in una somma di moltiplicazioni per potenze di due, siamo perfettamente in grado di calcolarla. L'algoritmo che utilizzeremo nella nostra routine di moltiplicazione generalizzata è il seguente: ruoteremo (raddoppieremo) uno dei due numeri (vi consigliamo il più grande), aggiungendo il risultato al totale ad ogni bit acceso all'interno dell'altro byte. Per semplificare, la nostra routine potrà considerare unicamente risultati minori di 255. Il seguente programma vi mostra come moltiplicare $1B per $09:

```
        LDA  #$09
        STA  $03
        LDA  #$1B
        STA  $04
        LDA  #$00
        ROR  $04
L70     ROL  $04
        LSR  $03
        BCC  L120
        CLC
        ADC  $04
```

```
L120 BNE  L70
     STA  $05
     BRK
```

Linee 1—6:	assegnano i valori da moltiplicare ed azzerano il totale (accumulatore). L'istruzione ROR seguita da ROL non ha nessun effetto la prima volta, ma solo quest'ultima fa parte del ciclo di calcolo.
Linea 7:	tranne la prima volta, provvede a raddoppiare uno dei due numeri ad ogni ciclo.
Linee 8—9:	spostano l'altro numero un bit alla volta nel flag di carry, testandone il contenuto per determinare se il primo numero deve essere o meno aggiunto al totale nel corrente ciclo. Se il carry è azzerato, la possibilità che il secondo numero sia stato completamente ruotato viene verificata dalla linea 12.
Linee 10—11:	sommano al totale (accumulatore) il numero raddoppiato ad ogni ciclo.
Linea 12:	se il salto condizionato della linea 9 è stato eseguito, essa verifica il termine della moltiplicazione (secondo numero completamente ruotato se uguale a zero). In caso contrario, questo BNE verrà sempre eseguito in quanto abbiamo appena addizionato un numero diverso da zero ad un totale che non oltrepasserà 255.
Linee 13—14:	fine dell'operazione; immagazzina il totale nella locazione $05.

Questa routine di moltiplicazione è decisamente più efficiente di quella accennata al capitolo 7. Con questo sistema, il numero massimo di cicli che dovremo effettuare è nove, ma nell'esempio presentato ne sono bastati solamente quattro (numero di bits necessari a formare il valore $09). Sostituendo nella prima e terza linea del programma i valori caricati nell'accumulatore, diventa possibile effettuare qualsiasi altra moltiplicazione (assicuratevi comunque a priori che il loro prodotto non oltrepassi mai 256).

Riassunto del Capitolo 10

[1] AND (Operazione di AND logico)

1 AND

	Ø	1
Ø	Ø	Ø
1	Ø	1

Spesso usata per mascherare bits da spegnere

[2] ORA (Operazione di OR logico)

2. ORA

	Ø	1
Ø	Ø	1
1	1	1

Spesso usata per mascherare bits da accendere

[3] EOR (Operazione di OR esclusivo logico)

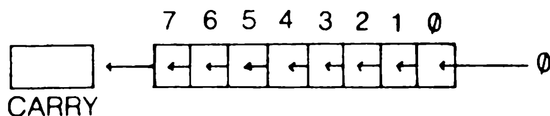
3. EOR (exclusive or)

	Ø	1
Ø	Ø	1
1	1	Ø

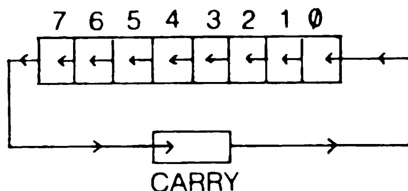
Spesso usata per mascherare bits da invertire

[4] BIT (AND logico senza memorizzazione del risultato)
Lo stato del flag di zero viene alterato. Il bit sei ed il bit sette del risultato vengono rispettivamente trasferiti nei flags di overflow e negativo.

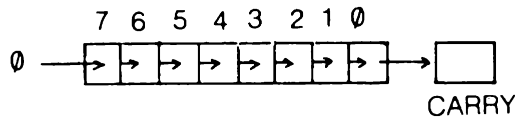
[5] ASL (Arithmetic Shift Left)



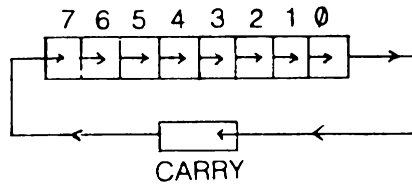
[6] ROL (Rotate One bit Left)



[7] LSR (Logical Shift Right)



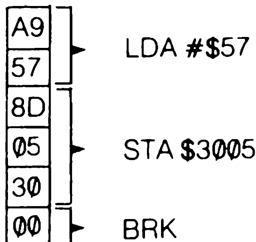
[8] ROR (Rotate One bit Right)



Particolari sul controllo del programma

Il contatore di programma

Ci siamo lungamente soffermati sulle varie operazioni che il microprocessore è in grado di svolgere, ma abbiamo detto poco o niente sul suo comportamento durante l'effettuazione di queste funzioni. È comunque giusto così, in quanto generalmente non abbiamo assolutamente bisogno di preoccuparcene. In particolari casi, tuttavia, venire a conoscenza di come il microprocessore opera ci permette di scoprire nuove istruzioni e potenti sue caratteristiche altrimenti nascoste.
Il microprocessore incorpora uno speciale registro di due bytes denominato "contatore di programma" (Program Counter), la cui unica funzione consiste nel tenere conto della posizione in memoria della successiva istruzione da eseguire. In altre parole, questo registro contiene l'indirizzo del successivo byte che deve essere caricato nel microprocessore e quindi utilizzato come istruzione.
Riconsideriamo l'analogia delle locazioni di memoria con le caselle postali. Ogni casella contiene un'istruzione od un dato del nostro programma, il quale ci appare, prendendo un esempio, sotto questa forma:



Per eseguirlo, è necessario prelevare il valore contenuto in ciascuna casella secondo l'ordine stabilito, e quindi agire di conseguenza. Supponiamo che tale ordine ci venga comunicato per mezzo di un contatore digitale, simile ad un orologio, che segnali l'indirizzo della casella interessata. Normalmente tale contatore dovrebbe incrementarsi di un'unità alla volta, stabilendo di prelevare il byte immediatamente successivo a quello appena trattato. Tuttavia, nel caso desideri farci spostare in una nuova

zona di memoria, non deve fare altro che comunicarci l'indirizzo della relativa locazione da prelevare; questo rappresenta esattamente il funzionamento dell'istruzione JMP. L'istruzione JMP $indirizzo si limita infatti a caricare lo specificato indirizzo di due bytes nel contatore di programma, trasferendo l'esecuzione all'istruzione in esso contenuta. I salti condizionati provvedono invece a sottrarre o addizionare al contatore di programma il valore in complemento a due ad essi associato, creando in tal modo un salto relativo (da cui il nome dell'indirizzamento utilizzato).

Il contatore di programma e le subroutines

Se fosse possibile immagazzinare il contenuto del contatore di programma immediatamente prima di un JMP, cambiandolo quindi con un nuovo indirizzo, avremmo più tardi la possibilità di ritornare alla stessa zona di memoria ricaricando nuovamente tale valore nel contatore di programma. Tutto questo rappresenta esattamente il concetto di subroutine. Un esempio BASIC di tale struttura è il seguente:

```
 10 PRINT"CIAO MONDO"
 20 GOSUB 100
 30 PRINT"IO STO BENE, GRAZIE"
 40 END
100 PRINT"COME STAI?"
110 RETURN
```

Sullo schermo verranno stampate nell'ordine le frasi:

```
CIAO MONDO
COME STAI?
IO STO BENE, GRAZIE
```

All'inizio del libro, avevamo definito un programma in linguaggio macchina come una forma particolare di subroutine richiamabile dal BASIC per mezzo del comando SYS. Similmente, anch'esso può contenere subroutines interne, richiamabili però attraverso una speciale istruzione denominata JSR (Jump to SubRoutine). Il ritorno da una subroutine viene provocato anche in questo caso per mezzo di RTS (ReTurn from Subroutine).

```
2000 LDX #$00      ┐
2002 JSR $2009     │
2005 INX           ├ PROGRAMMA PRINCIPALE
2006 BNE $2002     │
2008 BRK           ┘
```

```
2009 LDY #$03   ┐
200B STY $0C00  │
200E DEY        ├ SUBROUTINE
200F BNE $200B  │
2011 RTS        ┘
```

Ricordatevi che la velocità di esecuzione di questo programma è estremamente alta. L'impiego di subroutines costituisce un ottimo stile di programmazione per due motivi principali. Innanzitutto, diventa molto più facile localizzare e correggere eventuali errori all'interno di una subroutine, in quanto possono essere testati indipendentemente dal resto del programma. In secondo luogo, costruendo autonomamente una "biblioteca" di utili subroutines come cancellatori di schermo, ricercatori di bytes o trasferitori di zone di memoria, risulterà possibile e in qualunque momento aggiungere queste subroutines ai vostri programmi.
Abbiamo precisato come l'indirizzo di ritorno da una subroutine venga immagazzinato prima della sua chiamata, ma non abbiamo ancora specificato in che modo questo viene conservato. Esiste allo scopo un'apposita zona di memoria denominata "stack".

Struttura dello Stack

Flessibile e semplice da utilizzare, lo stack possiede una particolare caratteristica che permette la concatenazione di più subroutines una dentro l'altra. Esso non si limita pertanto a ricordare staticamente l'indirizzo di ritorno da ciascuna di esse, ma provvede a fornirlo al contatore di programma in ordine inverso a quello di chiamata. Lo stack è una cosiddetta struttura LIFO (Last In First Out), ovvero l'ultimo indirizzo ad entrarvi è anche il primo ad uscirne. Infatti, quando viene riscontrata un'istruzione RTS, è l'ultimo indirizzo di ritorno immagazzinato nello stack che deve essere trasferito nel contatore di programma allo scopo di ottenere un corretto proseguimento dell'esecuzione.
Riprendendo per l'ennesima volta il paragone con le caselle postali, supponiamo di scrivere per ogni chiamata di subroutine il rispettivo indirizzo di ritorno su un foglietto di carta che appoggiamo sopra il precedente. Lo stack rappresenta in questo caso il fermacarte. Al riscontro di un'istruzione RTS, il solo foglietto che siamo in grado di prelevare per conoscere l'indirizzo di ritorno è quello in cima al blocchetto, in quanto il fermacarte raccoglitore (stack) ci impedisce di accedere a quelli sottostanti. Tale foglietto superiore conterrà sempre il corretto indirizzo di ritorno, quello relativo all'ultima subroutine chiamata.

Le subroutines e lo Stack

Lo stack si trova localizzato in memoria dall'indirizzo $0100 all'indirizzo $01FF, occupando pertanto una pagina esatta (256 bytes). Esso viene progressivamente riempito in ordine decrescente, ma questo non altera minimamente il suo funzionamento. La cima dello stack (in realtà dovremmo dire la base) viene segnalata attraverso uno speciale registro denominato 'puntatore di stack' (Stack Pointer). Ogni volta che una subroutine viene chiamata per mezzo di un'istruzione JSR, l'attuale valore di due bytes del contatore di programma (PC) viene immagazzinato nello stack, mentre il suo puntatore (SP) viene decrementato di due unità.

PRIMA

CONTATORE DI PROGRAMMA	$AB	$CD

	STACK	INDIRIZZO
SP=XX	$JK	$1ØØ+XX

DOPO (JSR $PQMN)

CONTATORE DI PROGRAMMA	$PQ	$MN

	STACK	INDIRIZZO
	$JK	$1ØØ+XX
	$AB	$1ØØ+XX−1
SP=XX−2	$CD	$1ØØ+XX−2

Un'istruzione RTS effettua la procedura inversa. Essa preleva i due bytes immagazzinati in cima allo stack e li ritrasferisce nel contatore di programma. Similmente, il puntatore di stack viene incrementato di due unità.

PRIMA

CONTATORE DI PROGRAMMA	$PQ	MN

	STACK	INDIRIZZO
	$JK	$1ØØ+YY+2
	$AB	$1ØØ+YY+1
SP=YY	$CD	$1ØØ+YY

DOPO (RTS)

CONTATORE DI PROGRAMMA	$AB	$CD

	STACK	INDIRIZZO
SP=YY+2	$JK	$1ØØ+YY+2

Provate ad usare il comando DUMP da $100 a $200 per dare un'occhiata alla memoria di stack. Una grande caratteristica di flessibilità dello stack consiste nel poter immagazzinare e prelevare dati numerici per mezzo delle istruzioni PHA (PusH Accumulator onto the stack) e PLA (PuLl Accumulator off the stack), rispettivamente impiegate per trasferire il contenuto dell'accumulatore in cima allo stack e viceversa. Assicuratevi sempre che tali trasferimenti avvengano nell'ordine corretto. Utilizzando l'istruzione RTS dopo aver immagazzinato dati supplementari in cima allo stack, il suo indirizzo di ritorno verrà artificialmente determinato dagli ultimi due valori inseriti, qualunque essi siano. Vediamo un esempio:

```
2000  LDA #$0A
2002  PHA
2003  LDA #$20
2005  PHA
2006  RTS
2007  BRK
2008  BRK
2009  BRK
200A  LDA #$0F
200C  STA $0C00
200F  LDA #$0B
2011  STA $0C01
2014  BRK
```

Eseguendo questo programma, al riscontro dell'istruzione RTS verrà stampata sullo schermo la scritta OK.
Anche il puntatore di stack (SP) possiede particolari istruzioni di manipolazione; TSX provvede a trasferirne il contenuto nel registro X, mentre TXS effettua l'operazione inversa. Esercitatevi molto con queste ultime quattro istruzioni, in quanto un loro impiego scorretto pregiudicherebbe senza dubbio un corretto funzionamento dei vostri programmi.

Interrupts

Sebbene risulti sconsigliabile approfondire completamente il concetto di "interrupt" in un libro di questo genere, è tuttavia interessante illustrarne le caratteristiche di base unitamente al ruolo ricoperto nel sistema operativo del computer. Un interrupt (interruzione) viene generalmente inviato al microprocessore da un dispositivo esterno. Gli interrupts vengono principalmente utilizzati per avvertire il computer che qualcosa richiedente la sua attenzione sta avvenendo nel mondo esterno. Ad esempio, ogni sessantesimo di secondo viene inviato un interrupt per ricordare al calcolatore d'incrementare il clock interno e scandire la tastiera. Nel momento che un interrupt viene generato, il computer arresta immediatamente

quello che stava elaborando per saltare ad un'apposita routine di gestione. Al termine di tale routine, il controllo viene automaticamente restituito al punto dove è avvenuta l'interruzione. Esistono diversi tipi di interrupt, ognuno associato ad una particolare routine. Gli indirizzi iniziali di queste routines sono contenuti all'interno dei seguenti vettori di due bytes (indirizzamento indiretto):

$FFFE—$FFFF: questo vettore punta alla routine di gestione dei cosiddetti IRQ (Interrupt request). Questa particolare forma di interrupt viene generata ogni sessantesimo di secondo dal clock interno. Eventuali istruzioni BRK presenti in un programma rinviano anch'esse alla routine puntata da questo vettore. Interrupts di questo tipo possono essere disabilitati settando il flag di interrupt del registro di stato per mezzo dell'istruzione SEI, e reciprocamente riabilitati azzerando tale flag per mezzo dell'istruzione CLI. Per questo motivo, gli interrupts che passano attraverso questo vettore sono spesso definiti "mascherabili"

$FFFA—$FFFB: quest'altro vettore punta alla routine di gestione dei cosiddetti NMI (Non—maskable interrupt). Questa forma di interrupt non può essere disabilitata da programma. Il C16 non permette l'impiego di questi interrupts non mascherabili.

È possibile predisporre il computer in modo che al momento di andare ad eseguire la sua routine di gestione dell'interrupt IRQ, venga invece inviato ad un vostro programma. Questo è possibile grazie al fatto che esiste in memoria RAM un vettore analogo a quello visto in precedenza, situato in $0314—$0315. Esso può venire liberamente modificato in modo da farlo puntare alla vostra personale routine da gestire internamente all'interrupt. Vi raccomandiamo di non alterare avventatamente questo vettore senza avere prima consultato una documentazione specifica, altrimenti si potrebbe arrivare alla condizione di dover spegnere o resettare il computer per sbloccarlo.

Riassunto del Capitolo 11

[1] Il contatore di programma (PC) punta al successivo byte in memoria da utilizzare come istruzione.

[2] L'istruzione JMP immagazzina l'indirizzo nel PC.

[3] I salti condizionati aggiungono o sottraggono dal PC il valore ad essi associato.

[4] L'istruzione JSR immagazzina il PC nello stack ed il nuovo indirizzo nel PC (subroutine).

[5] L'istruzione RTS preleva i due bytes in cima allo stack e li immagazzina nel PC (indirizzo di ritorno).

[6] Lo stack può essere progressivamente riempito da una sola parte, e l'ultimo valore in esso immagazzinato è anche il primo ad essere eventualmente prelevato.

[7] Il registro puntatore di stack (SP) tiene conto della posizione della cima dello stack.
JSR incrementa SP di due.
RTS decrementa SP di due.

[8] Le istruzioni PHA e PLA immagazzinano e riprelevano l'accumulatore dallo stack. Assicuratevi sempre di prelevare valori dallo stack nell'ordine corretto.

[9] Le istruzioni TXS e TSX trasferiscono il contenuto del registro X nel puntatore di stack e viceversa.

[10] L'istruzione BRK provvede a trasferire nello stack il PC ed il registro di stato, immagazzinando nel PC l'indirizzo puntato dal vettore $FFFE–$FFFF.

[11] Le istruzioni PHP e PLP immagazzinano e riprelevano il registro di stato dallo stack.

[12] Gli interrupts vengono inviati al microprocessore da un dispositivo esterno. Essi trasferiscono il PC ed il registro di stato nello stack, immagazzinando nel PC l'indirizzo puntato dal vettore $FFFE–$FFFF. Vengono gestiti attraverso particolari routines ROM.

CAPITOLO 12

Il Kernal del Commodore 16

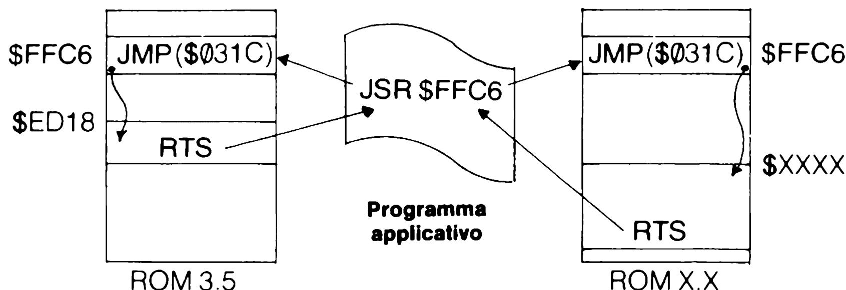
Concetti sul Kernal ed il sistema operativo

Un microprocessore, per quanto ampia possa essere la gamma delle sue istruzioni e grande la sua velocità esecutiva, non avrebbe alcuna possibilità di funzionare senza disporre di una serie di routines che provvedano alla sua supervisione. Questo programma di gestione e controllo del microprocessore è denominato "sistema operativo". Esso provvede ad accettare ciò che voi digitate sulla tastiera, lo visualizza, stampa un messaggio di errore nel caso non sia in grado d'interpretarlo, esegue il vostro comando se questo ha un senso, carica se necessario un programma dal disco oppure dal nastro, stampa qualcosa con la stampante se richiesto, ed esegue numerosissime altre funzioni che sarebbe troppo lungo elencare. Riassumendo, il sistema operativo coordina e gestisce tutte le risorse del computer, mettendo quest'ultimo al vostro servizio.

Il sistema operativo possiede una vasta collezione di routines che provvedono alla sua inizializzazione, alla gestione della memoria ed a tutte le procedure di input/output. Queste routines sono in generale strettamente dipendenti dall'hardware, ovvero ogni dispositivo dispone di sue proprie routines. Dal punto di vista dell'utente, è preferibile utilizzarle senza doversi preoccupare di quale dispositivo è in relazione con loro. La maggior parte dei costruttori di microcomputers prepara una lista delle routines di sistema richiamabili dal programmatore, accompagnate dai loro rispettivi indirizzi iniziali e relative tecniche d'impiego. I problemi nascono quando viene pubblicata una nuova versione aggiornata del sistema operativo; vi è infatti il rischio che le eventuali modifiche apportate ad alcune sue routines, generalmente accompagnate da un cambiamento dei loro indirizzi d'ingresso, oltre a creare difficoltà ai programmatori, rendano incompatibile il vecchio software facente uso di tali routines.

Nel COMMODORE 16 è stato risolto questo problema immagazzinando tutti gli indirizzi d'ingresso delle routines di supporto del sistema operativo in un'apposita tabella di salto (Jump Table) denominata KERNAL. Questa tabella di salto è locata nell'ultima pagina della memoria ROM. Essa contiene gli indirizzi iniziali di varie routines del sistema operativo immagazzinati in specifici vettori, i quali hanno il vantaggio di rimanere costanti in caso di modifiche. Essendo ogni routine richiamata attraverso

un salto indiretto, la sua effettiva posizione in memoria diventa a questo punto secondaria. Vediamo un grafico illustrativo:



Alcune utili routines del Kernal

Routine	Indirizzo	Funzione	Routines preparatorie	Registri di comunicazione	Registri interessati
		Aiuto al programmatore			
1. CHRIN	$FFCF	Ingresso di un carattere da tastiera	—	.A=carattere in ingresso	.X,.Y
2. CHROUT	$FFD2	Uscita di un carattere sullo schermo	—	.A=carattere in uscita	—
3. GETI	$FFE4	Prelevamento di un carattere dal buffer di tastiera	—	.A=Carattere prelevato =0 se nessuno	
4. PLOT	$FFF0	Legge/Assegna la posizi del cursore	—	Flag di carry: Lettura=1 Assegnamento=0 .X=Linea dello schermo (0-24) .Y=Colonna dello schermo (0-39)	
		Immagazzinamento I/O			
5. SETLFS	$FFBA	Assegna il numero logico del file Indirizzo primario (Canale della periferia) ed indirizzo secondario (Comando) della periferica	—	.A=Numero logico del file .X=Canale della periferica .Y=Comando =$FF se nessun comando	—
6. SETNAM	$FFBD	Assegna il nome del file	—	.A=Lunghezza del nome del file .X=Indirizzo del nome del file (byte basso) .Y=Indirizzo del nome de file (byte alto)	
7. LOAD	$FFD5	Carica/Verifica memori da una periferica	SETLFS SETNAM	.A=Load Verify=1	
8. SAVE	$FFD8	Registra memoria attraverso una periferica	SETLFS SETNAM	.A=Indirizzo di pagina dal puntatore d'inizio SAVE .X=Indirizzo del puntatore di fine SAVE (byte basso) .Y=Indirizzo del puntatore di fine SAVE (byte alto)	

Utilizzo delle routines del Kernal

Per utilizzare le routines del Kernal all'interno dei vostri programmi dovete:

<A> Determinare quella giusta da impiegare unitamente al suo relativo indirizzo d'ingresso.

<B> Chiamare le routines preparatorie, se necessario.

<C> Assegnare i parametri nei registri di comunicazione.

<D> Chiamare la routine.

<E> Trattare ogni errore di ritorno (indicato dal flag di carry settato).

<F> Salvare e ristabilire i registri interessati dalla routine, se necessario.

[1] CHRIN — Ingresso di un carattere da tastiera

Quando questa routine viene inizialmente chiamata, provvede a far lampeggiare il cursore e prelevare una linea di caratteri conclusa da un ritorno del carrello. Essa restituisce il codice del primo carattere nell'accumulatore. Le chiamate successive ritrovano uno alla volta tutti i caratteri precedentemente inseriti. Il ritrovamento di un ritorno del carrello determina la fine della linea. A questo punto, chiamandola nuovamente si ricadrà nelle condizioni iniziali.

[2] CHROUT — Uscita di un carattere sullo schermo

Viene stampato sullo schermo il carattere il cui codice ASCII è contenuto nell'accumulatore, e fatto avanzare il cursore di una posizione.

[3] GETIN — Prelevamento di un carattere dal buffer di tastiera

Ogni tasto premuto sulla tastiera viene intercettato dall'interrupt IRQ di sistema. Il suo codice ASCII viene immagazzinato in un apposito buffer che può contenere fino ad un massimo di dieci caratteri. Quando chiamata, questa routine preleva il primo carattere di tale buffer; se non ne trova nessuno, l'accumulatore viene caricato con il valore zero.

[4] PLOT — Legge/assegna la posizione del cursore

Questa routine è in grado di leggere o assegnare la posizione corrente del cursore sullo schermo in accordo con il flag di carry rispettivamente settato od azzerato. I registri X e Y contengono nell'ordine il numero della linea (0—24) e della colonna (0—39) costituenti le coordinate di schermo del cursore.

[5] SETLFS — Assegna il numero logico del file, nonché gli indirizzi primario e secondario della periferica utilizzata

Questa routine assegna il numero logico del file ad una periferica il cui canale sia compreso fra 0 e 31. Essa si occupa anche d'inviare un eventuale indirizzo secondario di comando. A seguire troverete un elenco dei canali riservati alle periferiche del C16:

Tastiera:	<0>
Cassetta:	<1>
Periferica RS—232:	<2>
Schermo:	<3>
Stampante seriale:	<4>
Disk drive seriale:	<8>

Nell'accumulatore viene immagazzinato il numero logico del file, in X il canale della periferica ed in Y l'eventuale indirizzo secondario. Se quest'ultimo non è richiesto, Y deve contenere il valore $FF.

[6] SETNAM — Assegna il nome del file

Questa routine assegna il nome del file da utilizzare con le routines di LOAD o SAVE. Nell'accumulatore viene immagazzinato il numero dei caratteri componenti il nome (lunghezza), mentre X e Y contengono rispettivamente byte basso e byte alto dell'indirizzo di memoria a partire dal quale si trova memorizzata la stringa del nome. Nel caso non venga richiesto alcun nome, è necessario immagazzinare il valore zero nell'accumulatore, ad indicare una lunghezza nulla.

[7] LOAD — Carica/verifica memoria da una periferica

Una volta chiamata con l'accumulatore posto a zero, questa routine provvede a caricare in memoria un file proveniente da un dispositivo

periferico. Nel caso invece venga chiamata con l'accumulatore posto a uno, essa verifica che un file proveniente da un dispositivo periferico sia identico al corrispondente contenuto della memoria.

[8] SAVE — Registra memoria attraverso una periferica

Questa routine registra una determinata porzione di memoria su un file indirizzato ad un dispositivo periferico. L'indirizzo iniziale dell'area da registrare è contenuto in un'apposito puntatore di pagina zero da immagazzinare nell'accumulatore. I registri X e Y devono infine contenere rispettivamente byte basso e byte alto dell'indirizzo finale dell'area da registrare.

Riassunto del Capitolo 12

[1] Il Kernal, contenuto nella memoria ROM, gestisce i contatti del computer con il mondo esterno.

[2] Le routines del Kernal saranno sempre compatibili con qualsiasi nuovo aggiornamento della memoria ROM.

Introduzioni alle Appendici

A seguire troverete utili ed interessanti tabelle informative necessarie per una buona programmazione del vostro COMMODORE 16. La loro funzione consiste principalmente in un supporto di riferimento, utilizzabile tanto dal principiante alle prime armi quanto dal programmatore esperto. Abbiamo inoltre allegato esaurienti spiegazioni ed occasionali esempi illustrativi a tutte le tabelle di più frequente consultazione. Quelle che invece non risultano dettagliatamente commentate escono dai limiti di questo libro e sono state incluse unicamente a scopo divulgativo. Vi consigliamo di considerare tutto quello che non siete momentaneamente in grado di comprendere come un punto di partenza verso la conoscenza di nozioni sempre più complesse e specifiche.

APPENDICE 1

Codici mnemonici del microprocessore 7501

Vi presentiamo una tabella riassuntiva comprendente tutte le istruzioni relative al microprocessore 7501, ognuna delle quali viene accompagnata da una descrizione, dai modi d'indirizzamento contemplati, dal formato utilizzato, dal numero di bytes occupati, dal codice esadecimale corrispondente ed infine da una lista dei flags del registro di stato conseguentemente alterati.

Istruzioni del microprocessore 7501 in ordine alfabetico

<ADC> Addiziona memoria ed accumulatore con riporto.
<AND> AND di memoria con accumulatore.
<ASL> Sposta un bit a sinistra (memoria od accumulatore).
<BCC> Salta con carry azzerato.
<BCS> Salta con carry settato.
<BEQ> Salta con risultato nullo.
<BIT> Testa i bits in memoria con accumulatore.
<BMI> Salta con risultato negativo.
<BNE> Salta con risultato non nullo.
<BPL> Salta con risultato positivo.
<BRK> Arresto forzato.
<BVC> Salta con overflow azzerato.
<BVS> Salta con overflow settato.
<CLC> Azzera il flag di carry.
<CLD> Disabilita il modo decimale.
<CLI> Azzera il flag disabilitatore di interrupt.
<CLV> Azzera il flag di overflow.
<CMP> Compara memoria ed accumulatore.
<CPX> Compara memoria e registro X.
<CPY> Compara memoria e registro Y.
<DEC> Decrementa memoria di un'unità.
<DEX> Decrementa il registro X di un'unità.
<DEY> Decrementa il registro Y di un'unità.
<EOR> OR esclusivo di memoria con accumulatore.
<INC> Incrementa memoria di un'unità.
<INX> Incrementa il registro X di un'unità.
<INY> Incrementa il registro Y di un'unità.
<JMP> Salta ad una nuova locazione.
<JSR> Salta ad una subroutine.

<LDA> Carica l'accumulatore con memoria.
<LDX> Carica il registro X con memoria.
<LDY> Carica il registro Y con memoria.
<LSR> Sposta un bit a destra (memoria od accumulatore).
<NOP> Nessuna operazione.
<ORA> OR di memoria con accumulatore.
<PHA> Salva l'accumulatore nello stack.
<PHP> Salva il registro di stato nello stack.
<PLA> Preleva l'accumulatore dallo stack.
<PLP> Preleva il registro di stato dallo stack.
<ROL> Ruota un bit a sinistra (memoria od accumulatore).
<ROR> Ruota un bit a destra (memoria od accumulatore).
<RTI> Ritorno da un interrupt.
<RTS> Ritorno da una subroutine.
<SBC> Sottrae memoria dall'accumulatore con prestito.
<SEC> Setta il flag di carry.
<SED> Abilita il modo decimale.
<SEI> Setta il flag disabilitatore di interrupt.
<STA> Immagazzina l'accumulatore in memoria.
<STX> Immagazzina il registro X in memoria.
<STY> Immagazzina il registro Y in memoria.
<TAX> Trasferisce l'accumulatore nel registro X.
<TAY> Trasferisce l'accumulatore nel registro Y.
<TSX> Trasferisce il puntatore di stack nel registro X.
<TXA> Trasferisce il registro X nell'accumulatore.
<TXS> Trasferisce il registro X nel puntatore di stack.
<TYA> Trasferisce il registro Y nell'accumulatore.

Codici mnemonici del microprocessore 7501

Nome Descrizione	Indirizzamento	Forma mnemonica	Numero di bytes	Codice esade-decimale	Registro di stato
ADC					N V - B D I ZC
Addiziona memoria	Immediato	ADC #Oper	2		●● ●●
ed accumulatore con riporto	Pagina Zero	ADC Oper	2		
	Pagina Zero.X	ADC Oper	2		
	Assoluto	ADC Oper	3		
	Assoluto.X	ADC Oper.X	3		
	Assuluto.Y	ADC Oper.Y	3		
	(Indiretto.X)	AND (Oper.X)	2		
	(indiretto).Y	ADC (Oper).Y	2		
AND					N V - B D I Z C
AND di memoria	Immediato	AND #Oper		29	● ●
con accumulatore	Pagina Zero	AND Oper		25	
	Pagina Zero.X	AND Oper.X		35	
	Assuluto	AND Oper		2D	
	Assoluto.X	AND Oper.X		3D	
	Assuluto.Y	AND Oper.Y		39	
	(Indiretto.X)	AND (Oper.X)		31	
	(Indiretto).Y	AND (Oper).Y		31	
ASL					N V - B D I Z C
Sposta un bit a sinistra	Accumulatore	ASL A	1	0A	● ●●
(memoria od accumulatore)	Pagina Zero	ASL Oper	2	06	
	Pagina Zero.X	2	16		
	Assoluto	ASL Oper	3	0E	
	Assoluto.X	ASL Oper.X	3	1E	
BCC					N V - B D I Z C
Salta con carry azzerato	Relativo	BCC Oper	2	90	
BCS					N V - B D I Z C
Salta con carry settato	Relativo	BCS Oper	2 B0		
BEQ					N V - B D I Z C
Salta con risultato nullo	Relativo	BEQ Oper	2	F0	
BIT					N V - B D I Z C
Testa i bits in memori	Pagina Zero	BIT Oper	2	24	MM ●
con accumulatore	Assoluto	BIT Oper	3	2C	7 6
BMI					N V - B D I Z C
Salta con risultato negativo	Relativo	BMI Oper	2	30	
BNE					N V - B D I Z C
Salta con risultato non nullo	Relativo	BNE Oper	2	D0	
BPL					N V - B D I Z C
Salta con risultato positivo	Relativo	BPL Oper	2	10	
BRK					N V - B D I Z C
Arresto forzato	Implicito	BRK	1	00	1 1
BVC					N V - B D I Z C
Salta con overflow azzerato	Relativo	BVC Oper	2	50	

Codici mnemonici del microprocessore 7501

Nome Descrizione	Indirizzamento	Forma mnemonica	Numero di bytes	Codice esadecimale	Registro di stato
BVS Salta con overflow settato	Relativo	BVS Oper	2	70	N V - B D I Z C
CLC Azzera il flag di carry	Implicito	CLC	1	18	N V - B D I Z C 0
CLD Disabilita il modo decimale	Implicito	CLD	1	D8	N V - B D I Z C 0
CLI Azzera il flag disabilitatore di interrupt	Implicito	CLI	1	58	N V - B D I Z C 0
CLV Azzera il flag overflow	Implicito	CLV	1	B8	N V - B D I Z C 0
CMP Compara memoria ad accumulatore	Immediato	CMP #Oper	2	C9	N V - B D I Z C ● ●●
	Pagina Zero	CMP Oper	2	C5	
	Pagina Zero.X	CMP Oper.X	2	D5	
	Assoluto	CMP Oper	3	CD	
	Assoluto.X	CMP Oper.X	3	DD	
	Assoluto.Y	CMP oper.Y	3	D9	
	(Indiretto.X)	CMP (Oper.X)	2	C1	
	(Indiretto).Y	CMP (Oper).Y	2	D1	
CPX Compara memori e registro X	Immediato	CPX #Oper	2	E0	N V - B D I Z C ● ●●
	Pagina Zero	CPX Oper	2	E4	
	Assoluto	CPX Oper	3	EC	
CPY Compara memoria e registro Y	Immediato	CPY #Oper	2	C0	N V - B D I Z C ● ●●
	Pagina Zero	CPY Oper	2	C4	
	Assoluto	CPY Oper	3	CC	
DEC Decrementa memori di un'unità	Pagina Zero	DEC Oper	2	C6	N V - B D I Z C ● ●
	Pagina Zero.X	DEC Oper.X	2	D6	
	Assoluto	DEC Oper	3	CE	
	Assoluto.X	DEC Oper.X	3	DE	
DEX Decrementa il registro X di un'unità		DEX			N V - B D I Z C ● ●
DEY Decrementa il registro Y di un'unità		DEY		88	N V - B D I Z C ● ●

Codici mnemonici del microprocessore 7501

Nome Descrizione	Indirizzamento	Forma mnemonica	Numero di bytes	Codice esade-cimale	Registro di stato
EOR					N V - B D I Z C
OR esclusivo di memori	Immediato	EOR #Oper	2	49	● ●
con accumulatore	Pagina Zero	EOR Oper	2	45	
	Pagina Zero.X	EOR Oper.X	2	55	
	Assoluto	EOR Oper	3	4D	
	Assoluto.X	EOR Oper.X	3	5D	
	Assoluto.Y	EOR Oper.Y	3	59	
	(Indiretto.X)	EOR (Oper.X)	2	41	
	(Indiretto).Y	EOR (Oper).Y	2	51	
INC					N V - B D I Z C
Incrementa memoria	Pagina Zero	INC Oper	2	E6	● ●
di un'unità	Pagina Zero.X	INC Oper.X	2	F6	
	Assoluto	INC Oper	3	EE	
	Assoluto.X	INC Oper.X	3	FE	
INX					N V - B D I Z C
Incrementa il reg. X di un'unità	Implicito	INX	1	E8	● ●
INY					N V - B D I Z C
Incrementa il reg. Y di un'unità	Implicito	INY	1	C8	● ●
JMP					N V - B D I Z C
Salta ad una nuova locazi	Assoluto	JMP Oper	3	4C	
	Indiretto	JMP (Oper)	3	6C	
JSR					N V - B D I Z C
Salta ad una	Assoluto	JSR Oper		20	
subroutine					
LDA					N V - B D I Z C
Carica l'accumulatore	Immediato	LDA #Oper	2		● ●
con memoria	Pagina Zero	LDA Oper	2		
	Pagina Zero.X	2	B5		
	Assoluto	LDA Oper	3	AD	
	Assoluto.X	LDA Oper.X	3	BD	
	Assoluto.Y	LDA Oper.Y	3	B9	
	(Indiretto.X)	LDA (Oper.X)	2	A1	
	(Indiretto).Y	LDA (Oper).Y	2	B1	
LDX					N V - B D I Z C
Carica il registro X	Immediato	LDX #Oper	2	A2	● ●
con memoria	Pagina Zero	LDX Oper	2	A6	
	Pagina Zero.Y	LDX Oper.Y	2	B6	
	Assoluto	LDX Oper	3	AE	
	Assoluto.Y	LDX Oper.Y	3	BE	
LDY					N V - B D I Z C
Carica il registro Y	Immediato	LDY #Oper	2	A0	● ●
con memoria	Pagina Zero	LDY Oper	2	A4	
	Pagina Zero.X	LDY Oper.X	2	B4	
	Assoluto	LDY Oper	3	AC	
	Assoluto.X	LDY Oper.X	3	BC	

Codici mnemonici del microprocessore 7501

Nome Descrizione	Indirizzamento	Forma mnemonica	Numero di bytes	Codice esade-cimale	Registro di stato
LSR					N V - B D I Z C
Sposta un bit a destra	Accumulatore	LSR A	1	4A	0 ●●
(memoria accumulatore)	Pagina Zero	LSR Oper	2	46	
	Pagina Zero.X	LSR Oper.X	2	56	
C → 7 8 6 4 3 2 1 0 → C	Assoluto	LSR Oper	3	4E	
	Assoluto.X	LSR Oper.X	3	5E	
NOP					N V - B D I Z C
Nessuna operazione	Implicito	NOP	1	EA	
ORA					N V - B D I Z C
OR di memoria	Immediato	ORA #Oper	2	09	● ●
con accumulatore	Pagina Zero	ORA Oper	2	05	
	Pagina Zero.X	ORA Oper.X	2	15	
	Assoluto	ORA Oper	3	0D	
	Assoluto.X	ORA Oper.X	3	1D	
	Assoluto.Y	ORA Oper.Y	3	19	
	(Indiretto.X)	ORA (Oper.X)	2	01	
	(Indiretto).Y	ORA (Oper).Y	2	11	
PHA					N V - B D I Z C
Salva l'accumulatore		PHA		48	
nello stack					
PHP					N V - B D I Z C
Salva il registro di	Implicito	PHP		08	
nello stack					
PLA					N V - B D I Z C
Preleva l'accumulatore	Implicito	PLA		68	● ●
dallo stack					
PLP					N V - B D I Z C
Preleva il registro di stato	Implicito	PLP		28	●●●●●●●●
dallo stack					
ROL					N V - B D I Z C
Ruota il bit a sinistra	Accumulatore	ROL A	1	2A	● ●●
(memoria od accumulatore)	Pagina Zero	ROL Oper	2	26	
	Pagina Zero.X	ROL Oper.	2	36	
7 6 5 4 3 2 1 0 ← C ←	Assoluto	ROL Oper	3	2E	
	Assoluto.X	3	3E		
ROR					N V - B D I Z C
Ruota il bit a destra	Accumulatore	ROR A	1	6A	● ●●
(memoria od accumulatore)	Pagina Zero	ROR Oper	2	66	
	Pagina Zero.X	ROR Oper.X	3	76	
C → 7 6 5 4 3 2 1 0 →	Assoluto	ROR Oper	3	6E	
	Assoluto.X	ROR Oper.X	3	7E	

Codici mnemonici del microprocessore 7501

Nome Descrizione	Indirizzamento	Forma mnemonica	Numero di bytes	Codice esade-cimale	Registro di stato
RTI Ritorno da interrupt	Implicito	RTI	1	40	N V - B D I Z C ● ● ● ● ● ● ● ●
RTS Ritorno da una subroutine	Implicito	RTS	1	60	N V - B D I Z C
SBC Sottrae memoria dall'accumulatore con prestito	Immediato Pagina Zero Pagina Zero.X Assoluto Assoluto.X Assoluto.Y (Indiretto.X) (Indiretto).Y	SBC #Oper SBC Oper SBC Oper.X SBC Oper SBC Oper.X SBC Oper.Y SBC (Oper.X) SBC (Oper).Y			N V - B D I Z C ●● ●●
SEC Setta il flag di carry	Implicito	SEC	1	38	N V - B D I Z C 1
SED Abilita il modo decimale	Implicito	SEC	1	F8	N V - B D I Z C 1
SEI Setta il flag disabilitatore di interrupt		SEI			N V - B D I Z C 1
STA Immagazzi in memori	Pagina Zero Pagina Zero.X Assoluto Assoluto.X Assoluto.Y (Indiretto.X) (Indiretto).Y	STA Oper STA Oper.X STA Oper STA Oper.X STA Oper.Y STA (Oper.X) STA (Oper).Y	2 2 3 3 3 2 2	85 95 8D 9D 99 81 91	N V - B D I Z C
STX Immagazzi il registro X in memori	Pagina Zero Pagina Zero.Y Assoluto	STX Oper STX Oper.Y STX Oper	2 2 3	86 96 8E	N V - B D I Z C
STY Immagazzi il registro Y in memori	Pagina Zero Pagina Zero.X Assoluto	STY Oper STY Oper.X STY Oper	2 2 3	84 94 8C	N V - B D I Z C
TAX Trasferisce l'accumulatore nel registro X		TAX			N V - B D I Z C ● ●
TAY Trasferisce l'accumulatore nel registro Y		TAY			N V - B D I Z C ● ●
TSX Trasferisce il puntatore di stack nel registro X	Implicito	TSX			N V - B D I Z C ● ●

Codici mnemonici del microprocessore 7501

Nome **Descrizione**	**Indirizzamento**	**Forma mnemonica**	**Numero di bytes**	**Codice esade-cimale**	**Registro di stato**
TXA Trasferisce il registro X nell'accumulatore		TXA			N V - B D I Z C ● ●
TXS Trasferisce il registro X nel puntatore di stack					N V - B D I Z C
TYA Trasferisce il registro Y nell'accumulatore				98	N V - B D I Z C ● ●

Codici del microprocessore 7501 in ordine numerico

00 — BRK
01 — ORA — (Indiretto.X)
02 — ???
03 — ???
04 — ???
05 — ORA — Pagina Zero
06 — ASL — Pagina Zero
07 — ???
08 — PHP
09 — ORA — Immediato
0A — ASL — Accumulatore
0B — ???
0C — ???
0D — ORA — Assoluto
0E — ASL — Assoluto
0F — ???
10 — BPL
11 — ORA — (Indiretto).Y
12 — ???
13 — ???
14 — ???
15 — ORA — Pagina Zero.X
16 — ASL — Pagina Zero.X
17 — ???
18 — CLC
19 — ORA — Assoluto.Y
1A — ???
1B — ???
1C — ???
1D — ORA — Assoluto.X
1E — ASL — Assoluto.X
1F — ???
20 — JSR
21 — AND — (Indiretto.X)
22 — ???
23 — ???
24 — BIT — Pagina Zero
25 — AND — Pagina Zero
26 — ROL — Pagina Zero
27 — ???
28 — PLP
29 — AND — Immediato
2A — ROL — Accumulatore
2B — ???
2C — BIT — Assoluto
2D — AND — Assoluto
2E — ROL — Assoluto
2F — ???
30 — BMI
31 — AND — (Indiretto).Y
32 — ???
33 — ???
34 — ???
35 — AND — Pagina Zero.X
36 — ROL — Pagina Zero.X
37 — ???
38 — SEC
39 — AND — Assoluto.Y
3A — ???
3B — ???
3C — ???
3D — AND — Assoluto.X
3E — ROL — Assoluto.X
3F — NOP
40 — RTI
41 — EOR — (Indiretto.X)
42 — ???
43 — ???
44 — ???
45 — EOR — Pagina Zero
46 — LSR — Pagina Zero
47 — ???
48 — PHA
49 — EOR — Immediato
4A — LSR — Accumulatore
4B — ???
4C — JMP — Assoluto
4D — EOR — Assoluto
4E — LSR — Assoluto
4F — ???
50 — BVC
51 — EOR (Indiretto).Y
52 — ???
53 — ???
54 — ???
55 — EOR — Pagina Zero.X
56 — LSR — Pagina Zero.X
57 — ???
58 — CLI
59 — EOR — Assoluto.Y
5A — ???
5B — ???
5C — ???
5D — EOR — Assoluto.X
5E — LSR — Assoluto.X
5F — ???
60 — RTS
61 — ADC — (Indiretto.X)
62 — ???
63 — ???
64 — ???
65 — ACD — Pagina Zero
66 — ROR — Pagina Zero
67 — ???
68 — PLA
69 — ADC — Immediato
6A — ROR — Accumulatore
6B — ???
6C — JMP — Indiretto
6D — ADC — Assoluto
6E — ROR — Assoluto
6F — ???
70 — BVS
71 — ADC — (Indiretto).Y
72 — ???
73 — ???
74 — ???
75 — ADC — Pagina Zero.X
76 — ROR — Pagina Zero.X
77 — ???
78 — SEI
79 — ADC — Assoluto.Y
7A — ???
7B — ???
7C — ???
7D — ADC — Assoluto.X
7E — ROR — Assoluto.X
7F — ???
80 — ???
81 — STA — (Indiretto.X)
82 — ???
83 — ???
84 — STY — Pagina Zero
85 — STA — Pagina Zero
86 — STX — Pagina Zero
87 — ???
88 — DEY
89 — ???
8A — TXA
8B — ???
8C — STY — Assoluto

Codici del microprocessore 7501 in ordine numerico

8D — STA — Assoluto
8E — STX — Assoluto
8F — ???
90 — BCC
91 — STA — (Indiretto).Y
92 — ???
93 — ???
94 — STY — Pagina Zero.X
95 — STA — Pagina Zero.X
96 — STX — Pagina Zero.Y
97 — ???
98 — TYA
99 — STA — Assoluto.Y
9A — TXS
9B — ???
9C — ???
9D — STA — Assoluto.X
9E — ???
9F — ???
A0 — LDY — Immediato
A1 — LDA — (Indiretto.X)
A2 — LDX — Immediato
A3 — ???
A4 — LDY — Pagina Zero
A5 — LDA — Pagina Zero
A6 — LDX — Pagina Zero
A7 — ???
A8 — TAY
A9 — LDA — Immediato
AA — TAX
AB — ???
AC — LDY — Assoluto
AD — LDA — Assoluto
AE — LDX — Assoluto
AF — ???
B0 — BCS
B1 — LDA — (Indretto).Y
B2 — ???
B3 — ???
84 — LDY — Pagina Zero.X
B5 — LDA — Pagina Zero.X
B6 — LDX — Pagina Zero.Y
B7 — ???
B8 — CLV
B9 — LDA — Assoluto.Y
BA — TSX
BB — ???
BC — LDY — Assoluto.X
BD — LDA — Assoluto.X
BE — LDX — Assoluto.Y
BF — ???
C0 — CPY — Immediato
C1 — CMP — (Indiretto.X)
C2 — ???
C3 — ???
C4 — CPY — Pagina Zero
C5 — CMP — Pagina Zero
C6 — DEC — Pagina Zero
C7 — ???
C8 — INY
C9 — CMP — Immediato
CA — DEX
CB — ???
CC — CPY — Assoluto
CD — CMP — Assoluto
CE — DEC — Assoluto
CF — ???
D0 — BNE
C1 — CMP — (Indiretto).Y
D2 — ???
D3 — ???
D4 — ???
D5 — CMP — Pagina Zero.X
D6 — DEC — Pagina Zero.X
D7 — ???
D8 — CLD
D9 — CMP — Assoluto.Y
DA — ???
D8 — ???
DC — ???
DD — CMP — Assoluto.X
DE — DEC — Assoluto.X
DF —
E0 — CPX — Immediato
E1 — SBC — (Indiretto.X)
E2 — ???
E3 — ???
E4 — CPX — Pagina Zero
E5 — SBC — Pagina Zero
E6 — INC — Pagina Zero
E7 — ???
E8 — INX
E9 — SBC — Immediato
EA — NOP
EB — ???
EC — CPX — Assoluto
ED — SBC — Assoluto
EE — INC — Assoluto
EF — ???
F0 — BEQ
F1 — SBC — (Indiretto).Y
F2 — ???
F3 — ???
F4 — ???
F5 — SBC — Pagina Zero.X
F6 — INC — Pagina Zero.X
F7 — ???
F8 — SED
F9 SBC — Assoluto.Y
FA — ???
FB — ???
FC — ???
FD — SBC — Assoluto.X
FE — INC — Assoluto.X
FF — ???

??? Operazione indefinita

Registri del microprocessore 7501

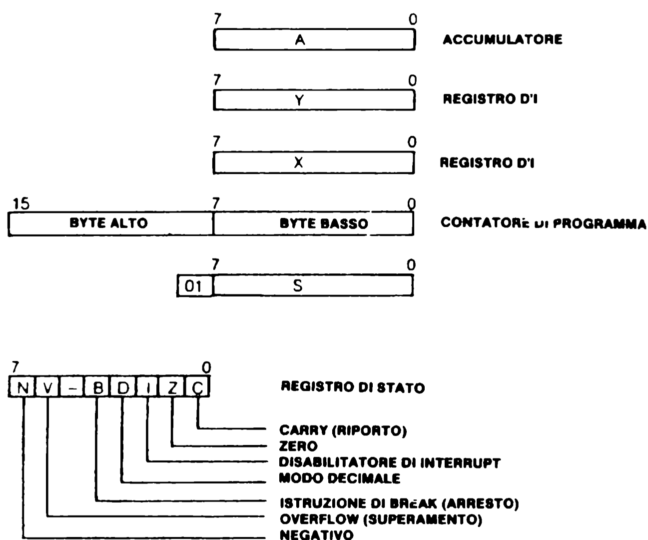


Tavola di conversione da esadecimale a decimale
Cifra meno significativa

Cifra più significativa

HEX	0		1		2		3		4		5		6		7	
	Byte basso	Byte alto	Byte basso	Byte alto	Byte basso	Byte alto	Byte basso	Byte alto	Byte basso	Byte alto	Byte basso	Byte alto	Byte basso	Byte alto	Byte basso	Byte alto
0	0	0	1	256	2	512	3	768	4	1024	5	1280	6	1536	7	1792
1	16	4096	17	4352	18	4608	19	4864	20	5120	21	5376	22	5632	23	5888
2	32	8192	33	8448	34	8704	35	8960	36	9216	37	9472	38	9728	39	9984
3	48	12288	49	12544	50	12800	51	13056	52	13312	53	13568	54	13824	55	14080
4	64	16384	65	16640	66	16896	67	17152	68	17408	69	17664	70	17920	71	18176
5	80	20480	81	20736	82	20992	83	21248	84	21504	85	21760	86	22016	87	22272
6	96	24576	97	24832	98	25088	99	25344	100	25600	101	25856	102	26112	103	26368
7	112	28672	113	28928	114	29184	115	29440	116	29696	117	29952	118	30208	119	30464
8	128	32768	129	33024	130	33280	131	33536	132	33792	133	34048	134	34304	135	34560
9	144	36864	145	37120	146	37376	147	37632	148	37888	149	38144	150	38400	151	38656
A	160	40960	161	41216	162	41472	163	41728	164	41984	165	42240	166	42496	167	42752
B	176	45056	177	45312	178	45568	179	45824	180	46080	181	46336	182	46592	183	46848
C	192	49152	193	49408	194	49664	195	49920	196	51076	197	50432	198	50688	199	50944
D	208	53248	209	53504	210	53760	211	54016	212	54272	213	54528	214	54784	215	55040
E	224	57344	225	57600	226	57856	227	58112	228	58368	229	58624	230	58880	231	59136
F	240	61440	241	61696	242	61952	243	62208	244	62464	245	62720	246	62976	247	63232

HEX	8		9		A		B		C		D		E		F	
	Byte basso	Byte alto	Byte basso	Byte alto	Byte basso	Byte alto	Byte basso	Byte alto	Byte basso	Byte alto	Byte basso	Byte alto	Byte basso	Byte alto	Byte basso	Byte alto
0	8	2048	9	2304	10	2560	11	2816	12	3072	13	3328	14	3584	15	3840
1	24	6144	25	6400	26	6656	27	6912	28	7168	29	7424	30	7680	31	7936
2	40	10240	41	10496	42	10752	43	11008	44	11264	45	11520	46	11776	47	12032
3	56	14336	57	14592	58	14848	59	15704	60	15360	61	15616	62	15872	63	16128
4	72	18432	73	18688	74	18944	75	19200	76	19456	77	19712	78	19968	79	20224
5	88	22528	89	22784	90	23040	91	23296	92	23552	93	23808	94	24064	95	24320
6	104	26624	105	26880	106	27136	107	27392	108	27648	109	27904	110	28160	111	28416
7	120	30720	121	30976	122	31232	123	31488	124	31744	125	32000	126	32256	127	32512
8	136	34816	137	35072	138	35328	139	35584	140	35840	141	36096	142	36352	143	36608
9	152	38912	153	39168	154	39424	155	39680	156	39936	157	40192	158	40448	159	40704
A	168	43008	169	43264	170	43520	171	43776	172	44032	173	44288	174	44544	175	44800
B	184	47104	185	47360	186	47616	187	47872	188	48128	189	48384	190	48640	191	48896
C	200	51200	201	51456	202	51712	203	51968	204	52224	205	52480	206	52736	207	52992
D	216	55296	217	55552	218	55808	219	56064	220	56320	221	56576	222	56832	223	57088
E	232	59392	233	59648	234	59904	235	60160	236	60416	237	60672	238	60928	239	61184
F	248	63488	249	63744	250	64000	251	64256	252	64512	253	64768	254	65024	255	65280

APPENDICE 3

Tavola di conversione da esadecimale a decimale

Questa tabella permette di convertire in decimale numeri esadecimali composti da un massimo di quattro cifre. Vediamo come utilizzarla:

[1] Dividete il numero da convertire in gruppi di due cifre. Ad esempio:
$F17B —> F1 7B
$2A —> 2A

[2] Prendete il suo byte basso (7B o 2A nell'esempio precedente). Ricercate la cifra più significativa (7) nella colonna di sinistra e la cifra meno significativa (B) nella linea superiore. A questo punto trovate la casella d'intersezione della linea (7) con la colonna (B). Essa contiene due numeri (123 e 31488). Tali numeri corrispondono al valore decimale di 7B rispettivamente considerato come byte basso o byte alto. Dato che ci stiamo occupando del byte basso, prendete il valore 123. Ripetete la stessa operazione con il byte alto (F1), prendendo questa volta il valore di destra 61696. L'equivalente decimale del numero $F17B si ottiene infine sommando i due valori 123 e 61696 (123 + 61696 = 61819).

Per convertire numeri esadecimali di due cifre, è necessario considerarli come bytes bassi, prelevando quindi dalle caselle il valore di sinistra. Numeri composti da una o tre cifre devono essere preceduti da uno zero, in modo da trasformarli rispettivamente in numeri di due o quattro cifre.

APPENDICE 4

Calcolo dell'indirizzo di un salto condizionato
Tavole di numerazione in complemento a due

[1] Per calcolare un salto relativo, contate il numero di bytes compresi fra la locazione immediatamente successiva all'istruzione di salto ed il suo indirizzo di destinazione. Se quest'ultimo precede tale locazione, utilizzate la tabella di salto all'indietro; se invece lo segue, la tabella da consultare diventa quella di salto in avanti. Ricercate adesso tale valore nella giusta tabella, ricavando le cifre bassa ed alta dell'intervallo di salto rispettivamente in corrispondenza della riga e colonna esterna. Queste tabelle possono anche essere utilizzate per il procedimento inverso, ricavando l'intervallo a partire dalle cifre che lo compongono.

[2] Per convertire un numero decimale munito di segno compreso fra −128 e +127 in un numero esadecimale in complemento a due, ricercatelo innanzitutto nella tabella corrispondente al suo segno. Leggete quindi le cifre bassa ed alta del numero in complemento a due rispettivamente in corrispondenza della riga e colonna esterna. Il procedimento inverso si ottiene ricercando nella giusta tabella il valore decimale corrispondente alle cifre di partenza del numero in complemento a due da convertire, ricordandosi infine di farlo precedere dal segno meno se negativo.

Tavole di salto relativo e di notazione in complemento a due

SALTO RELATIVO IN AVANTI — NUMERI POSITIVI

ALTA \ BASSA	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

SALTO RELATIVO ALL'INDIETRO — NUMERI NEGATIVI

ALTA \ BASSA	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
9	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
A	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81
B	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
C	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
D	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
E	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
F	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Mappa di memoria dettagliata del Commodore 16

Etichetta	*Indirizzi Esadecimale*	*Decimale*	*Descrizione*
PDIR	$0000	0	Registro direzione dati 7501
PORT	$0001	1	Registro I/O 7501 a 8 bits
SRCHTK	$0002	2	Token di ricerca
ZPVEC1	$0003–0004	3–4	Temporaneo (renumber)
ZPVEC2	$0005–0006	5–6	Temporaneo (renumber)
CHARAC	$0007	7	Carattere di ricerca
ENDCHR	$0008	8	Flag: apici a fine stringa
TRMPOS	$0009	9	Colonna di schermo dell'ultima TAB
VERCK	$000A	10	Flag: 0=LOAD 1=VERIFY
COUNT	$000B	11	Puntatore input buffer/Nr. elementi
DIMFLG	$000C	12	Flag: Dimensionamento array per difetto
VALTYP	$000D	13	Tipo dati: $FF=stringa $00=numeri
INTFLG	$000E	14	Tipo dati: $80=interi $00=virgola mobile
DORES	$000F	15	Flag: scansione DATA/indice LIST
SUBFLG	$0010	16	Flag: chiamata funzione utente/rif. indice
INPFLG	$0011	17	Flag: $00=INPUT $40=GET $98=READ
TANSGN	$0012	18	Flag: segno di TAN/confronto risultato
CHANNL	$0013	19	Flag: sollecito di input
LINNUM	$0014–0015	20–21	Temporaneo: valore intero
TEMPPT	$0016	22	Puntatore: stack temporaneo stringhe
LASTPT	$0017–0018	23–24	Ultimo indirizzo temporaneo stringa
TEMPST	$0019–0021	25–33	Stack temporaneo di stringhe
INDEX1	$0022–0023	34–35	Puntatore area utilità
INDEX2	$0024–0025	36–37	Puntatore area utilità
RESHO	$0026	38	
RESMOH	$0027	39	
RESMO	$0028	40	
RESLO	$0029	41	
	$002A	42	
TXTTAB	$002B–002C	43–44	Puntatore inizio testo BASIC
VARTAB	$002D–002E	45–46	Puntatore inizio variabile BASIC
ARYTAB	$002F–0030	47–48	Puntatore inizio arrays BASIC
STREND	$0031–0032	49–50	Puntatore fine arrays BASIC (+1)
FRETOP	$0033–0034	51–52	Puntatore fondo memoria stringhe
FRESPC	$0035–0036	53–54	Puntatore stringa utilità
MEMSIZ	$0037–0038	55–56	Puntatore: massimo indirizzo BASIC
CURLIN	$0039–003A	57–58	Corrente numero di linea BASIC
TXTPTR	$003B–003C	59–60	Precedente numero di linea BASIC
FNDPNT	$003D–003E	61–62	

Etichetta	Indirizzi Esadecimale	Decimale	Descrizione
DATLIN	$003F–0040	63–64	Corrente numero di linea DATA
DATPTR	$0041–0042	65–66	Puntatore: corrente indirizzo elemento DATA
INPPTR	$0043–0044	67–68	Vettore: routine di INPUT
VARNAM	$0045–0046	69–70	Corrente nome di variabile BASIC
VARPNT	$0047–0048	71–72	Puntatore: corrente dato variabile BASIC
FORPNT	$0049–004A	73–74	Puntatore: indice di FOR/NEXT
OPPTR	$004B–004C	75–76	
OPMASK	$004D	77	
DEFPNT	$004E–004F	78–79	
DSCPNT	$0050–0051	80–81	
	$0052	82	
HELPER	$0053	83	
JMPER	$0054	84	
SIZE	$0055	85	
OLDOV	$0056	86	
TEMPF1	$0057	87	
HIGHDS	$0058–0059	88–89	
HIGHTR	$005A–005B	90–91	
	$005C	92	
LOWDS	$005D–005E	93–94	
LOWTR	$005F	95	
EXPSGN	$0060	96	
FACEXP	$0061	97	Acc. Virgola mobile #1: esponente
FACHO	$0062	98	Acc. Virgola mobile #1: mantissa
FACMOH	$0063	99	
FACMO	$0064	100	
FACLO	$0065	101	
FACSGN	$0066	102	Acc. Virgola mobile #1: segno
SGNFLG	$0067	103	Puntatore: valutazione permanenza segno
BITS	$0068	104	Acc. Virgola mobile #1: overflow
ARGEXP	$0069	105	Acc. Virgola mobile #2: esponente
ARGHO	$006A	106	Acc. Virgola mobile #2: mantissa
ARGMOH	$006B	107	
ARGMO	$006C	108	
ARGLO	$006D	109	
ARGSGN	$006E	110	Acc. Virgola mobile #2: segno
ARISGN	$006F	111	Segno comparazione #1 con #2
FACOV	$0070	112	Accumulatore mobile #1: rango basso
FBUFPT	$0071–0072	113–114	Puntatore: buffer di cassetta
AUTINC	$0073–0074	115–116	Valore incremento per AUTO (0=off)
MVDFLG	$0075	117	Flag: 10 Kbytes Hi–res allocati
KEYNUM	$0076	118	
KEYSIZ	$0077	119	
SYNTMP	$0078	120	Temporaneo: caricamenti indiretti
DSDESC	$0079–007B	121–123	Descrittore per ds$
TOS	$007C–007D	124–125	Cima stack in fase di esecuzione

Etichetta	Indirizzi Esadecimale	Decimale	Descrizione
TMPTON	$007E—007F	126—127	Temporaneo: tono e volume musica
VOICNO	$0080	128	
RUNMOD	$0081	129	
POINT	$0082	130	
GRAPHM	$0083	131	Corrente modo grafico
COLSEL	$0084	132	Corrente colore prescelto
MC1	$0085	133	Multicolor numero 1
FGC	$0086	134	Colore principale
SCXMAX	$0087	135	Numero massimo di colonne
SCYMAX	$0088	136	Numero massimo di linee
LTFLAG	$0089	137	Flag di colorazione a sinistra
RTFLAG	$008A	138	Flag di colorazione a destra
STOPNB	$008B	139	Stop colorazione se non c'é colore sfondo
GRAPNT	$008C—008D	140—141	
VTEMP1	$008E	142	
VTEMP2	$008F	143	
STATUS	$0090	144	Byte di stato del Kernal: ST
STKEY	$0091	145	Flag: tasto STOP/tasto RVS
SPVERR	$0092	146	Temporaneo
VERFCK	$0093	147	Flag: 0=LOAD 1=VERIFY
C3P0	$0094	148	Flag: carattere in uscita-bus seriale
BFOUR	$0095	149	Carattere bufferizzato
XSAV	$0096	150	Temporaneo per BASIC
LDTND	$0097	151	Numero di files aperti
DFLTN	$0098	152	Canale d'ingresso per difetto (0)
DFLTO	$0099	153	Canale d'uscita (CMD) per difetto (3)
MSGFLG	$009A	154	Flag: $80=diretto $00=programma
SAL	$009B	155	Marchio di errore nastro passo #1
SAH	$009C	156	Marchio di errore nastro passo #2
EAL	$009D	157	
EAH	$009E	158	
T1	$009F—00A0	159—160	Area dati temporanea
T2	$00A1—00A2	161—162	Area dati temporanea
TIME	$00A3—00A5	163—165	Clock in tempo reale 1/60 secondo
R2D2	$00A6	166	Utilizzo del bus seriale
TPBYTE	$00A7	167	Byte da leggere/scrivere su nastro
BSOUR1	$00A8	168	Usato dalla routine seriale
FPVERR	$00A9	169	
DCOUNT	$00AA	170	
FNLEN	$00AB	171	Lunghezza corrente nome del file
LA	$00AC	172	Corrente numero logico del file
SA	$00AD	173	Corrente indirizzo secondario
FA	$00AE	174	Corrente canale del dispositivo
FILDR	$00AF—00B0	175—176	Puntatore: corrente nome del file
ERRSUM	$00B1	177	
STAL	$00B2	178	Indirizzo iniziale I/O (byte basso)
STAH	$00B3	179	Indirizzo iniziale I/O (byte alto)
MEMUSS	$00B4—00B5	180—181	Caricamento base della RAM

Etichetta	Indirizzi Esadecimale	Decimale	Descrizione
TAPEBS	$00B6—00B7	182—183	Puntatore: base della cassetta
TMP2	$00B8—00B9	184—185	
WRBASE	$00BA—00BB	186—187	Puntatore: dato scritto su nastro
IMPARM	$00BC—00BD	188—189	Puntatore: stringa immediata
FETPTR	$00BE—00BF	190—191	Puntatore: prelevamento di banchi
SEDSAL	$00C0—00C1	192—193	Temporaneo per scrolling
RVS	$00C2	194	Flag acceso del modo RVS
INDX	$00C3	95	
LSXP	$00C4	196	Posizione X alla partenza
LSTP	$00C5	197	
SFDX	$00C6	198	Flag: modo shift per stampa
CRSW	$00C7	199	Flag: INPUT o GET da tastiera
PNT	$00C8—00C9	200—201	Puntatore corrente ind. linea di schermo
PNTR	$00CA	202	Colonna cursore nella corrente linea
QTSW	$00CB	203	Flag: editor con apici ($00=no)
SEDT1	$00CC	204	Editor temporaneamente in uso
TBLX	$00CD	205	Corrente numero linea del cursore
DATAX	$00CE	206	Area dati temporanea
INSRT	$00CF	207	Flag: modo insert, > 0=nr. inserzioni
	$00D0—00D7	208—215	Area usata da software parlante
	$00D8—00E8	216—232	Area usata da software applicativo
CIRSEG	$00E9	233	Editor di unione linee di schermo
USER	$00EA—00EB	234—235	Editor del colore di schermo
KEYTAB	$00EC—00ED	236—237	Tabella indiretta scansione tastiera
TMPKEY	$00EE	238	
NDX	$00EF	239	Indice coda buffer di tastiera
STPFLG	$00F0	240	Flag di pausa
TO	$00F1—00F2	241—242	Monitor di stoccaggio in pagina zero
CHRPTR	$00F3	243	
BUFEND	$00F4	244	
CHKSUM	$00F5	245	Temporaneo: calcolo del checksum
LENGHT	$00F6	246	
PASS	$00F7	247	Passaggio in esecuzione
TYPE	$00F8	248	Tipo del blocco
USEKDY	$00F9	249	Bit 7=1 scrittura; Bit 6=1 lettura
XSTOP	$00FA	250	Salva xreq per rapido test di STOP
CURBNK	$00FB	251	Corrente configurazione dei banchi
XON	$00FC	252	Carattere da inviare per X—on
XOFF	$00FD	253	Carattere da inviare per X—off
SEDT2	$00FE	254	Editor temporaneamente in uso
LOFBUF	$00FF	255	
FBUFFR	$0100—010F	256—271	
SAVEA	$0110	272	Locazioni temporanee per:
SAVEX	$0111	273	Save
SAVEY	$0112	274	Restore
COLKEY	$0113—0122	275—289	Tabella RAM luminosità/colori
SYSSTK	$0124—01FF	291—511	Stack del sistema operativo
BUF	$0200—0258	512—600	BASIC/monitor buffer
OLDLIN	$0259—025A	601—602	memoria BASIC

Etichetta	Indirizzi Esadecimale	Decimale	Descrizione
OLDTXT	$025B–025C	603–604	memoria BASIC
	$025D–02AC	605–684	Area d'interfacciamento BASIC/DOS
XCNT	$025D	605	Contatore di ciclo del DOS
FNBUFR	$025E–026D	606–621	Immagazzinamento nome del file
DOSF1L	$026E	622	Lunghezza nome del file #1
DOSDS1	$026F	623	Disk drive #1
DOSF1A	$0270–0271	624–625	Indirizzo nome del file #1
DOSF2L	$0272	626	Lunghezza nome del file #2
DOSDS2	$0273	627	Disk drive #2
DOSF2A	$0274–0275	628–629	Indirizzo nome del file #2
DOSLA	$0276	630	Indirizzo logico del DOS
DOSFA	$0277	631	Indirizzo fisico del DOS
DOSSA	$0278	632	Indirizzo secondario del DOS
DOSDID	$0279–027A	633–634	Identificatore dischetto del DOS
DIDCHK	$027B	635	Flag DID del DOS
DOSSTR	$027C	636	Buffer di uscita stringhe del DOS
DOSSPC	$027D–02AC	637–684	Area di costruzione stringhe DOS

AREA UTILIZZATA DALLE ROUTINES GRAFICHE

XPOS	$02AD–02AE	685–686	Corrente posizione orizzontale
YPOS	$02AF–02B0	687–688	Corrente posizione verticale
XDEST	$02B1–02B2	689–690	Coordinata orizzontale di arrivo
YDEST	$02B3–02B4	691–692	Coordinata verticale di arrivo
XABS	$02B5–02B6	693–694	
YABS	$02B7–02B8	695–696	
XSGN	$02B9–02BA	697–698	
YSGN	$02BB–02BC	699–700	
FCT1	$02BD–02BE	701–702	
FCT2	$02BF–02C0	703–704	
ERRVAL	$02C1–02C2	705–706	
LESSER	$02C3	707	
GREATR	$02C4	708	
ANGSGN	$02C5	709	Segno dell'angolo
SINVAL	$02C6–02C7	710–711	Seno dell'angolo
COSVAL	$02C8–02C9	712–713	Coseno dell'angolo
ANGCNT	$02CA–02CB	714–715	Angolo/distanza temporaneo

INIZIO ZONA A DEFINIZIONE MULTIPLA #1

	$02CC	716	Indicatore di posizione
BNR	$02CD	717	Puntatore: numero iniziale
ENR	$02CE	718	Puntatore: numero finale
DOLR	$02CF	719	Flag di dollaro
FLAG	$02D0	720	Flag di virgola
SWE	$02D1	721	Contatore
USGN	$02D2	722	Segno dell'esponente
UEXP	$02D3	723	Puntatore all'esponente

Etichetta	*Indirizzi* *Esadecimale*	*Decimale*	*Descrizione*
VN	$02D4	724	Numero di cifre prima della virgola
CHSN	$02D5	725	Flag di giustificazione
VF	$02D6	726	nr. cifre significative pre—virgola
NF	$02D7	727	nr. cifre significative post—virgola
POSP	$02D8	728	Flag di segno (campo)
FESP	$02D9	729	Flag di esponente (campo)
ETOF	$02DA	730	Interrutore
CFORM	$02DB	731	Contatore di carattere (campo)
SNO	$02DC	732	Numero di segni
BLFD	$02DD	733	Campo vuoto/asterisco
BEGFD	$02DE	734	Puntatore: inizio del campo
LFOR	$02DF	735	Lunghezza del formato
ENDFD	$02E0	736	Puntatore: fine del campo

INIZIO ZONA A DEFINIZIONE MULTIPLA #2

XCENTR	$02CC—02CD	716—717	
YCENTR	$02CE—02CF	718—719	
XDIST1	$02D0—02D1	720—721	
YDIST1	$02D2—02D3	722—723	
XDIST2	$02D4—02D5	724—725	
YDIST2	$02D6—02D7	726—727	
	$02D8—02D9	728—729	Indicatore di posizione
COLCNT	$02DA	730	Contatore: colonna del carattere
ROWCNT	$02DB	731	Contatore: riga del carattere
STRCNT	$02DC	732	

INIZIO ZONA A DEFINIZIONE MULTIPLA #3

XCORD1	$02CC—02CD	716—717	
YCORD1	$02CE—02CF	718—719	
BOXANG	$02D0—02D1	720—721	Angolo di rotazione
XCOUNT	$02D2—02D3	722—723	
YCOUNT	$02D4—02D5	724—725	
BXLENG	$02D6—02D7	726—727	Lunghezza di un lato
XCORD2	$02D8—02D9	728—729	
YCORD2	$02DA—02DB	730—731	
XCIRCL	$02CC—02CD	716—717	Coordinata X del centro del cerchio
YCIRCL	$02CE—02CF	718—719	Coordinata Y del centro del cerchio
XRADUS	$02D0—02D1	720—721	Raggio X
YRADUS	$02D2—02D3	722—723	Raggio Y
ROTANG	$02D4—02D5	724—725	Angolo di rotazione
	$02D6—02D7	726—727	
ANGBEG	$02D8—02D9	728—729	Inizio dell'arco dell'angolo
ANGEND	$02DA—02DB	730—731	Fine dell'arco dell'angolo
XRCOS	$02DC—02DD	732—733	Raggio X * cos (angolo rotazione)
YRSIN	$02DE—02DF	734—735	Raggio Y * sin (angolo rotazione)
XRSIN	$02E0—02E1	736—737	Raggio X * sin (angolo rotazione)
YRCOS	$02E2—02E3	738—739	Raggio Y * cos (angolo rotazione)

Etichetta	Indirizzi Esadecimale	Decimale	Descrizione
			INIZIO ZONA A DEFINIZIONE MULTIPLA #4
	$02CC	716	Indicatore di posizione
KEYLEN	$02CD	717	
KEYNXT	$02CE	718	
STRSZ	$02CF	719	Lunghezza stringa
GETTYP	$02D0	720	Modo di sostituzione stringa
STRPTR	$02D1	721	Contatore: posizione della stringa
OLDBYT	$02D2	722	Vecchio byte di alta risoluzione
NEWBYT	$02D3	723	Nuovo byte di alta risoluzione
	$02D4	724	Indicatore di posizione
XSIZE	$02D5–02D6	725–726	Lunghezza colonna della figura
YSIZE	$02D7–02D8	727–728	Lunghezza riga della figura
XSAVE	$02D9–02DA	729–730	Temporaneo: lunghezza della colonna
STRADR	$02DB–02DC	731–732	Salva descrittore stringa della figura
BITIDX	$02DD	733	Indice del bit all'interno del byte
SAVSIZ	$02DE–02E1	734–737	Stoccaggio temporaneo di lavoro
	$02E2–02E3	738–739	
CHRPAG	$02E4	740	Byte alto indirizzo ROM caratteri
BITCNT	$02E5	741	Temporaneo per GSHAPE
SCALEM	$02E6	742	Flag: modo scala
WIDTH	$02E7	743	Flag: doppia larghezza
FILFLG	$02E8	744	Flag: riempimento rettangoli
BITMSK	$02E9	745	Temporaneo per mascheramento bits
NUMCNT	$02EA	746	
TRCFLG	$02EB	747	Flags: modo TRACE
T3	$02EC	748	
T4	$02ED–02EE	749–750	
VTEMP3	$02EF	751	Temporaneo: stoccaggio grafica
VTEMP4	$02F0	752	
VTEMP5	$02F1	753	
ADRAY1	$02F2–02F3	754–755	Vettore: converte decimale in intero
ADRAY2	$02F4–02F5	756–757	Vettore: converte intero in decimale
	$02F6–02FD	758–765	
BNKVEC	$02FE–02FF	766–767	Vettore: cartucce di funzione
IERROR	$0300–0301	768–769	Errore indiretto (uscita in X)
IMAIN	$0302–0303	770–771	Main indiretto (ciclo di sistema)
ICRNCH	$0304–0305	772–773	Crunch indiretto (tokenizzazione)
IQPLOP	$0306–0307	774–775	List indiretto (elenco caratteri)
IGONE	$0308–0309	776–777	Gone indiretto (invio caratteri)
IEVAL	$030A–030B	778–779	Eval indiretto (valutazione simboli)
IESCLK	$030C–030D	780–781	Uscita dalla tokenizzazione
IESCPR	$030E–030F	782–783	
IESCEX	$0310–0311	784–785	
ITIME	$0312–0313	786–787	
CINV	$0314–0315	788–789	Vettore RAM di interrupt IRQ
CBINV	$0316–0317	790–791	Vettore RAM dell'istruzione BRK
IOPEN	$0318–0319	792–793	Vettore: routine OPEN

Etichetta	Indirizzi Esadecimale	Decimale	Descrizione
ICLOSE	$031A–031B	794–795	Vettore: routine CLOSE
ICHKIN	$031C–031D	796–797	Vettore: routine CHKIN
ICKOUT	$031E–031F	798–799	Vettore: routine CHKOUT
ICLRCH	$0320–0321	800–801	Vettore: routine CLRCHN
IBASIN	$0322–0323	802–803	Vettore: routine CHRIN
IBSOUT	$0324–0325	804–805	Vettore: routine CHROUT
ISTOP	$0326–0327	806–807	Vettore: routine STOP
IGETIN	$0328–0329	808–809	Vettore: routine GETIN
ICLALL	$032A–032B	810–811	Vettore: routine CLALL
USRCMD	$032C–032D	812–813	Vettore definibile dall'utente
ILOAD	$032E–032F	814–815	Vettore: routine LOAD
ISAVE	$0330–0331	816–817	Vettore: routine SAVE
	$0332	818	
TAPBUF	$0333–03F2	819–1010	Buffer di cassetta
WRLEN	$03F3–03F4	1011–1012	Lunghezza dati da scrivere su nastro
RDCNT	$03F5–03F6	1013–1014	Lunghezza dati da leggere dal nastro
INPQUE	$03F7–0436	1015–1078	Coda input dell'RS–232
ESTAKL	$0437–0454	1079–1108	
ESTAKH	$0455–0472	1109–1138	
CHRGET	$0473–0478	1139–1144	
CHRGOT	$0479–0484	1145–1156	
QNUM	$0485–0493	1157–1171	
INDSUB	$0494–04A1	1172–1185	Subroutine prelevamento ROM divisa
ZERO	$04A2–04A4	1186–1188	Costante numerica per il BASIC
INDTXT	$04A5–04AF	1189–1199	Puntatore di testo
INDIN1	$04B0–04BA	1200–1210	Indice & indice 1
INDIN2	$04BB–04C5	1211–1221	Indice 2
INDST1	$04C6–04D0	1222–1232	Stringa 1
INDLOW	$04D1–04DB	1233–1243	
INDFMO	$04DC–04E6	1244–1254	
PUFILL	$04E7	1255	Stampa usando il simbolo di FILL
PUCOMA	$04E8	1256	Stampa usando la virgola
PUDOT	$04E9	1257	Stampa usando il simbolo di periodo
PUMONY	$04EA	1258	Stampa usando il segno dollaro
TMPDES	$04EB–04EE	1259–1262	Temporaneo per istruzioni
ERRNUM	$04EF	1263	Ultimo codice di errore
ERRLIN	$04F0–04F1	1264–1265	Numero di linea dell'ultimo errore
TRAPNO	$04F2–04F3	1266–1267	Linea destinazione in caso di errore
TMPTRP	$04F4	1268	Temporaneo: contiene numero di TRAP
ERRTXT	$04F5–04F6	1269–1270	
OLDSTK	$04F7	1271	
TMPTXT	$04F8–04F9	1272–1273	
TMPLIN	$04FA–04FB	1274–1275	
MTIMLO	$04FC–04FD	1276–1277	Tavola salti da effettuare
MYIMHI	$04FE–04FF	1278–1279	
USRPOK	$0500–0502	1280–1282	
RNDX	$0503–0507	1283–1287	
DEJAVU	$0508	1288	Stato di partenza "a freddo" o "calda"

Etichetta	Indirizzi Esadecimale	Decimale	Descrizione
LAT	$0509—0512	1289—1298	Numeri logici dei files
FAT	$0513—051C	1299—1308	Numeri primari canali dispositivi
SAT	$051D—0526	1309—1318	Indirizzi secondari
KEYD	$0527—0530	1319—1328	Buffer di tastiera IRQ
MEMSTR	$0531—0532	1329—1330	Inizio della memoria
MSIZ	$0533—0534	1331—1332	Cima della memoria
TIMOUT	$0535	1333	Flag IEEE di sospensione
FILEND	$0536	1334	Flag di fine file: 1=raggiunto 2=no
CTALLY	$0537	1335	Numero caratteri rimasti nel buffer
CBUFVA	$0538	1336	Totale caratteri validi nel buffer
TPTR	$0539	1337	Puntatore carattere seguente nel buffer
FLTYPE	$053A	1338	Corrente tipo del file su nastro
COLOR	$053B	1339	Byte attivo di attributo
FLASH	$053C	1340	Flag: carattere lampeggiante
	$053D	1341	Byte Libero!!!
HIBASE	$053E	1342	Locazione base della cima di schermo
XMZX	$053F	1343	
RPTFLG	$0540	1344	Flag: ripetizione tasti
KOUNT	$0541	1345	
DELAY	$0542	1346	
SHFLAG	$0543	1347	Flag: tasto SHIFT
LSTSHF	$0544	1348	Ultimo esempio shiftato
KEYLOG	$0545—0546	1349—1350	Assegnamento tabella di tastiera
MODE	$0547	1351	
AUTODN	$0548	1352	Flag: autoscroll in basso (0=on)
LINTMP	$0549	1353	
ROLFLG	$054A	1354	
FORMAT	$054B	1355	Stoccaggio monitor fuori pagina zero
	$054C—054E	1356—1358	
WRAP	$054F	1359	
TMPC	$0550	1360	
DIFF	$0551	1361	
PCH	$0552	1362	Contatore di programma byte alto
PCL	$0553	1363	Contatore di programma byte basso
FLGS	$0554	1364	Immagine del registro di stato
ACC	$0555	1365	Immagine dell'accumulatore
XR	$0556	1366	Immagine del registro X
YR	$0557	1367	Immagine del registro Y
SP	$0558	1368	Immagine del puntatore di stack
INVL	$0559	1369	
INVH	$055A	1370	
CMPFLG	$055B	1371	Usata da varie routines del monitor
BAD	$055C	1372	
KEYIDX	$055D	1373	Usata per tasti programmabili
KEYDIX	$055E	1374	
KEYBUF	$055F—0566	1375—1382	Tabella lunghezze di P.F.
PKYBUF	$0567—05E6	1383—1510	Area immagazzinamento tasti P.F.
KDATA	$05E7	1511	Temporaneo: scrittura dati kennedy

Etichetta	Indirizzi Esadecimale	Decimale	Descrizione
KDYCMD	$05E8	1512	Kennedy in lettura o scrittura
KDYNUM	$05E9	1513	Numero canale del kennedy
KDYPRS	$05EA	1514	Flag: $FF=kennedy presente $00=no
KDYTYP	$05EB	1515	Temporaneo: tipo OPEN per kennedy
SAVRAM	$05EC–06EB	1516–1771	Pagina usata da routines di banking
PAT	$05EC–05EF	1516–1519	Tabella degli indirizzi fisici
LNGJMP	$05F0–05F1	1520–1521	Indirizzo di salto lungo
FETARG	$05F2	1522	Accumulatore di salto lungo
FETXRG	$05F3	1523	Registro X di salto lungo
FETSRG	$05F4	1524	Registro di stato di salto lungo
AREAS	$05F5–065D	1525–1629	Zone di RAM per banking
ASPECH	$065E–06EB	1630–1771	Zona di RAM per sintesi vocale
STKTOP	$06EC–07AF	1772–1967	Stack dell'esecuzione BASIC
WROUT	$07B0	1968	Byte da scrivere su nastro
PARITY	$07B1	1969	Temporaneo per calcoli di parità
TT1	$07B2	1970	Temporaneo per scrittura header
TT2	$07B3	1971	Temporaneo per scrittura header
	$07B4	1972	
RDBITS	$07B5	1973	Indice locale della routine READBYTE
ERRSP	$07B6	1974	Puntatore nello stack di errore
FPERRS	$07B7	1975	Numero errori del primo passaggio
DSAMP1	$07B8–07B9	1976–1977	Costante di tempo
DSAMP2	$07BA–07BB	1978–1979	Costante di tempo
ZCELL	$07BC–07BD	1980–1981	Costante di tempo
SRECOV	$07BE	1982	Marcatore di stack ripresa stopkey
DRECOV	$07BF	1983	Marcatore di stack ripresa dropkey
TRSAVE	$07C0–07C3	1984–1987	Parametri passati a RDBLOK
RDSTMP	$07C4	1988	Stato temporaneo salvato per RDBLOK
LDRSCN	$07C5	1989	Abbreviazioni consecutive in leader
CDERRM	$07C6	1990	Errori fatali in conto alla rovescia
VSAVE	$07C7	1991	Temporaneo per comando VERIFY
T1PIPE	$07C8–07CB	1992–1995	Condotto temporaneo per T1
ENEXT	$07CC	1996	Diffusione di errore in lettura

SEZIONE DI MEMORIA PER RS–232

UOUTQ	$07CD	1997	Carattere dell'utente da inviare
UOUTFG	$07CE	1998	Flag: 0=buffer vuoto 1=pieno
SOUTQ	$07CF	1999	Carattere di sistema da inviare
SOUNFG	$07D0	2000	Flag: 0=buffer vuoto 1=pieno
INQFPT	$07D1	2001	Puntatore: inizio della coda input
INQRPT	$07D2	2002	Puntatore: termine della coda input
INQCNT	$07D3	2003	Numero caratteri in coda di input
ASTAT	$07D4	2004	Stato temporaneo per ACIA
AINTMP	$07D5	2005	Temporaneo per routine di input
ALSTOP	$07D6	2006	Flag per pausa locale
ARSTOP	$07D7	2007	Flag per pausa distante
APRES	$07D8	2008	Flag: 0=no ACIA 1=ACIA
KLUDES	$07D9–07E4	2009–2020	Routine indiretta sottocaricata

Etichetta	*Indirizzi* *Esadecimale*	*Decimale*	*Descrizione*
SCBOT	$07E5	2021	
SCTOP	$07E6	2022	
SCLF	$07E7	2023	
SCRT	$07E8	2024	
SCRDIS	$07E9	2025	
INSFLG	$07EA	2026	
LSTCHR	$07EB	2027	
LOGSCR	$07EC	2028	
TCOLOR	$07ED	2029	
BITABL	$07EE–07F1	2030–2033	

IMMAGAZZINAMENTO REGISTRI DURANTE UNA SYS

SAREG	$07F2	2034	Accumulatore
SXREG	$07F3	2035	Registro X
SXREG	$07F4	2036	Registro Y
SPREG	$07F5	2037	Contatore di programma
LSTX	$07F6	2038	Indice di scansione della tastiera
STPDSB	$07F7	2039	Flag: disabilita pausa CONTROL – S
RAMROM	$07F8	2040	MSB prelievi monitor: 0=ROM 1=RAM
COLSW	$07F9	2041	MSB tabella color/lum: 0=RAM 1=ROM
FERMSK	$07FB	2043	Maschera VM per divisione schermo
LSEM	$07FC	2044	Semaforo arresto motore per nastro
PALCNT	$07FD	2045	PAL
	$07FE–07FF	2046–2047	
TEDATR	$0800–0BFF	2048–3071	Bytes attributi di colore dello schermo
TEDSCN	$0C00–0FFF	3072–4095	Puntatori ai caratteri dello schermo
BASBGN	$1000–	4096–	Inizio area del testo BASIC
GRBASE	$2000–	8192–	Inizio BASIC con alta risoluzione
BMLUM	$1800–1BFF	6144–7167	Tabella di luminosità hi–res
BMCOLR	$1C00–1FFF	7168–8191	Tabella dei colori hi–res
CHRBAS	000–D7FF	53248–55295	Inizio di 2K ROM dei caratteri
	800–FCFF	55296–64767	KERNAL ROM

BANKING JUMP TABLE

$FCF1	64753	JMP alla routine IRQ della cartuccia
$FCF4	64756	JMP alla routine PHOENIX
$FCF7	64759	JMP alla routine LONG FETCH
$FCFA	64762	JMP alla routine LONG JUMP
$FCFD	64765	JMP alla routine LONG IRQ

JUMP TABLE UFFICIOSA

$FF9	65353	JMP alla routine definizione tasti
$FF4C	65356	JMP alla routine PRINT
$FF4F	65359	JMP alla routine PRIMM

Etichetta	Indirizzi Esadecimale	Decimale	Descrizione
	$FF52	65362	JMP alla routine ENTRY
	$FF80	65408	Numero del KERNAL: bit più significativo 0=NTSC 1=PAL

JUMP TABLE DEL KERNAL

CINT	$FF81	65409	Inizializza l'editor di schermo
IOINIT	$FF84	65412	Inizializza i dispositivi I/O
RAMTAS	$FF87	65415	RAM test
RESTOR	$FF8A	65418	Ripristina valori iniziali ai vettori
VECTOR	$FF8D	65421	Cambia vettori per l'utente
SETMSG	$FF90	65424	Messaggi di controllo del sistema
SECND	$FF93	65427	Invia SA dopo LISTEN
TKSA	$FF96	65430	Invia SA dopo TALK
MEMTOP	$FF99	65433	Assegna/legge cima della memoria
MEMBOT	$FF9C	65436	Assegna/legge base della memoria
SCNKEY	$FF9F	65439	Scansione della tastiera
SETTMO	$FFA2	65442	Assegna sospensione sul disco DMA
ACPTR	$FFA5	65445	Byte in entrata dal bus seriale (Haudshake)
CIOUT	$FFA8	65448	Byte in uscita dal bus seriale (Haudshake)
UNTLK	$FFAB	65451	Invia UNTALK al bus seriale
UNLSN	$FFAE	65454	Invia UNLISTEN al bus seriale
LISTN	$FFB1	65457	Invia LISTEN al bus seriale
TALK	$FFB4	65460	Invia TALK al bus seriale
READSS	$FFB7	65463	Restituisce il byte di stato I/O
SETLFS	$FFBA	65466	Assegna i parametri del file logico
SETNAM	$FFBD	65469	Assegna lunghezza e indirizzo nome file
OPEN	$FFC0	65472	Apre un file logico
CLOSE	$FFC3	65475	Chiude un file logico
CHKIN	$FFC6	65478	Apre un canale in entrata
CHOUT	$FFC9	65481	Apre un canale in uscita
CLRCH	$FFCC	65484	Chiude tutti i canali I/O
BASIN	$FFCF	65487	Input da un canale
BSOUT	$FFD2	65490	Uscita attraverso un canale
LOADSP	$FFD5	65493	Carica da un file
SAVESP	$FFD8	65496	Salva su un file
SETTIM	$FFDB	65499	Assegna il clock interno
RDTIM	$FFDE	65502	Legge il clock interno
STOP	$FFE1	65505	Verifica la pressione del tasto STOP
GETIN	$FFE4	65508	Preleva un carattere dal buffer
CLALL	$FFE7	65511	Chiude tutti i files
UDTIM	$FFEA	65514	Incrementa il clock
SCRORG	$FFED	65517	Organizzazione dello schermo
PLOT	$FFF0	65520	Legge/assegna coordinate del cursore
IOBASE	$FFF3	65523	Restituisce locazione inizio I/O

Mappa dei registri del Ted Chip

Hex	Reg	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
$FF00	0	Timer 1 Bit 7	Timer 1 Bit 6	Timer 1 Bit 5	Timer 1 Bit 4	Timer 1 Bit 3	Timer 1 Bit 2	Timer 1 Bit 1	Timer 1 Bit 0
$FF01	1	Timer 1 Bit 15	Timer 1 Bit 14	Timer 1 Bit 13	Timer 1 Bit 12	Timer 1 Bit 11	Timer 1 Bit 10	Timer 1 Bit 9	Timer 1 Bit 8
$FF02	2	Timer 2 Bit 7	Timer 2 Bit 6	Timer 2 Bit 5	Timer 2 Bit 4	Timer 2 Bit 3	Timer 2 Bit 2	Timer 2 Bit 1	Timer 2 Bit 0
$FF03	3	Timer 2 Bit 15	Timer 2 Bit 14	Timer 2 Bit 13	Timer 2 Bit 12	Timer 2 Bit 11	Timer 2 Bit 10	Timer 2 Bit 9	Timer 2 Bit 8
$FF04	4	Timer 3 Bit 7	Timer 3 Bit 6	Timer 3 Bit 5	Timer 3 Bit 4	Timer 3 Bit 3	Timer 3 Bit 2	Timer 3 Bit 1	Timer 3 Bit 0
$FF05	5	Timer 3 Bit 15	Timer 3 Bit 14	Timer 3 Bit 13	Timer 3 Bit 12	Timer 3 Bit 11	Timer 3 Bit 10	Timer 3 Bit 9	Timer 3 Bit 8
$FF06	6	Test	Extend color	Alta risoluzione	Disabilita lo schermo	24/25 linee	Vert. Scroll 2	Vert. Scroll 1	Vert. Scroll 0
$FF07	7	Reverse spento	PAL/ NTSC	Congela-tore	Multi color	39/40 colonne	Horz. Scroll 2	Horz. Scroll 1	Horz. Scroll 0
$FF08	8	ALLACCIAMENTO TASTIERA							
$FF09	9	Request Intrupt	Timer 3 Intrupt	N/C	Timer 2 Intrupt	Timer 1 Intrupt	Lt. pen Intrupt	Raster Intrupt	N/C
$FF0A	10	N/C	Abilita T3. Int	N/C	Abilita T2. Int	Abilita T1 Int	Abilita LP. Int	Abilita Raster	Raster Comp. 8
$FF0B	11	Raster Comp. 7	Raster Comp. 6	Raster Comp. 5	Raster Comp. 4	Raster Comp. 3	Raster Comp. 2	Raster Comp. 1	Raster Comp. 0
$FF0C	12	N/C	N/C	N/C	N/C	N/C	N/C	Cursore Bit 9	Cursore Bit 8
$FF0D	13	Cursore Bit 7	Cursore Bit 6	Cursore Bit 5	Cursore Bit 4	Cursore Bit 3	Cursore Bit 2	Cursore Bit 1	Cursore Bit 0
$FF0E	14	Voce 1 Bit 7	Voce 1 Bit 6	Voce 1 Bit 5	Voce 1 Bit 4	Voce 1 Bit 3	Voce 1 Bit 2	Voce 1 Bit 1	Voce 1 Bit 0
$FF0F	15	Voce 2 Bit 7	Voce 2 Bit 6	Voce 2 Bit 5	Voce 2 Bit 4	Voce 2 Bit 3	Voce 2 Bit 2	Voce 2 Bit 1	Voce 2 Bit 0

Hex	Reg	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
$FF10	16	N/C	N/C	N/C	N/C	N/C	N/C	Voce 2 Bit 9	Voce 2 Bit 8
$FF11	17	Ricarica suono	Voce 2 Rum. b.	Voce 2 Selezione	Voce 1 Selezione	Volume Bit 3	Volume Bit 2	Volume Bit 1	Volume Bit 0
$FF12	18	N/C	N/C	Alta risoluzione base 2	Alta risoluzione base 1	Alta risoluzione base 0	Rom/Ram Selezione	Voce 1 Bit 9	Voce 1 Bit 8
$FF13	19	Carattere base 5	Carattere base 4	Carattere base 3	Carattere base 2	Carattere base 1	Carattere base 0	Singolo clock	Stato
$FF14	20	Matrice video 4	Matrice video 3	Matrice video 2	Matrice video 1	Matrice video 0	N/C	N/C	N/C
$FF15	21	N/C	Sfondo Lum 2	Sfondo Lum 1	Sfondo Lum 0	Sfondo Col 3	Sfondo Col 2	Sfondo Col 1	Sfondo Col 0
$FF16	22	N/C	Carattere Lum 2	Carattere Lum 1	Carattere Lum 0	Carattere Col 3	Carattere Col 2	Carattere Col 1	Carattere Col 0
$FF17	23	N/C	Multi Lum 2	Multi Lum 1	Multi Lum 0	Multi Col 3	Multi Col 2	Multi Col 1	Multi Col 0
$FF18	24	N/C	Multi Lum 2	Multi Lum 1	Multi Lum 0	Multi Col 3	Multi Col 2	Multi Col 1	Multi Col 0
$FF19	25	N/C	Cornice Lum 2	Cornice Lum 1	Cornice Lum 0	Cornice Col 3	Cornice Col 2	Cornice Col 1	Cornice Col 0
$FF1A	26	N/C	N/C	N/C	N/C	N/C	N/C	Ricarica alta risol.	Ricarica alta risol.
$FF1B	27	Ricar. alta risoluz. 7	Ricar. alta risoluz. 6	Ricar. alta risoluz. 5	Ricar. alta risoluz. 4	Ricar. alta risoluz. 3	Ricar. alta risoluz. 2	Ricar. alta risoluz. 1	Ricar. alta risoluz. 0
$FF1C	28	N/C	N/C	N/C	N/C	N/C	N/C	N/C	Linea verticale 8
$FF1D	29	Linea verticale 7	Linea verticale 6	Linea verticale 5	Linea verticale 4	Linea verticale 3	Linea verticale 2	Linea verticale 1	Linea verticale 0
$FF1E	30	Posizione orizz. 8	Posizione orizz. 7	Posizione orizz. 6	Posizione orizz. 5	Posizione orizz. 4	Posizione orizz. 3	Posizione orizz. 2	Posizione orizz. 1
$FF1F	31	N/C	Bit di lampeggio 3	Bit di lampeggio 2	Bit di lampeggio 1	Bit di lampeggio 0	Subindiriz. verticale 2	Subindiriz. verticale 1	Subindiriz. verticale 0
$FF3E	62	SELEZIONE ROM							
$FF3F	63	SELEZIONE RAM							

Descrizione dei registri del Ted Chip

Registri da #0 a #5: Timers interni

Il TED chip possiede tre timers interni a 16 bits. Ognuno di essi è fisicamente suddiviso in due registri di 8 bits occupanti due successive locazioni di memoria. I timers vengono decrementati ad una determinata frequenza, 884 KHz nel sistema PAL e 894 KHz nel sistema NTSC, generando un interrupt al momento in cui raggiungono lo zero. Essi devono essere inizializzati attraverso la seguente procedura:

[A] Disabilitare qualsiasi forma di interrupt.
[B] Assegnare il byte basso del timer.
[C] Assegnare il byte alto del timer.
[D] Abilitare gli interrupts desiderati.

È fondamentale che non trascorrano più di 125 microsecondi fra l'assegnamento del byte basso e quello del byte alto, altrimenti si ricadrà in una condizione di errore relativa al conteggio del timer.

Il timer #1 comprende i registri 0 (byte basso) e 1 (byte alto). Scrivendo in questi registri si determina il valore di ricarica del timer, ovvero il massimo valore a partire dal quale inizia la fase di decremento verso lo zero. In quel momento, viene emesso un interrupt ed il ciclo riprende dal suddetto valore di ricarica.

I timers #2 e #3 rappresentano dei contatori a corsa libera. Decrementandosi fino allo zero, essi riprendono quindi il conteggio da $FFFF. Possono essere letti o modificati in qualsiasi momento.

Registro #6: Formato dello schermo

I bits 0—2 di questo registro determinano la posizione verticale di scorrimento (scrolling). Il bit 3 commuta fra 24 (=0) e 25 (=1) linee di schermo. Per eseguire uno scrolling verticale, il bit 3 deve essere azzerato, ed i bits 0—2 incrementati o decrementati in funzione della direzione desiderata. Nel caso non venga richiesto alcuno scrolling verticale, è necessario settare il bit 3 e fare in modo che i bits 0—2 rappresentino il valore $03.

Il bit 4 corrisponde al bit disabilitatore di schermo. Nel caso venga settato, lo schermo viene gestito e visualizzato normalmente. Se invece viene azzerato, lo schermo e tutti i prelevamenti del TED chip vengono disabilitati, permettendo così al microprocessore di procedere ad una velocità

esecutiva quasi doppia (1,768 MHz nel sistema PAL e 1,788 MHz nel sistema NTSC).

Il bit 5 ed il bit 6 inseriscono rispettivamente il modo grafico in alta risoluzione ed il modo del colore esteso quando vengono posti ad uno. Il bit 7 viene infine utilizzato come verifica del chip, e deve costantemente rimanere azzerato.

Registro #7: Formato dello schermo

I bits 0—2 di questo registro determinano la posizione orizzontale di scorrimento (scrolling). Il bit 3 commuta fra 38 (=0) e 40 (=1) colonne di schermo. Per eseguire uno scrolling orizzontale, il bit 3 deve essere azzerato; incrementando i bits 0—2 lo schermo verrà spostato verso destra, mentre decrementandoli lo spostamento avverrà verso sinistra. Nel caso non si desideri alcuno scrolling orizzontale, i bits 0—2 devono essere tutti azzerati.

Il bit 4 inserisce il modo multicolore quando viene settato. Il bit 5 è il cosiddetto "bit congelatore"; esso quando settato, inibisce il TED chip ad incrementare le posizioni verticale ed orizzontale, nonchè i timers. Il bit 6 seleziona il sistema video PAL se posto ad uno, oppure il sistema video NTSC se azzerato.

Il bit 7 rappresenta il disabilitatore del modo reverse. Normalmente tale bit viene posto a zero, mettendo così a disposizione 128 configurazioni di caratteri. Ogni carattere può essere invertito (visualizzato in negativo) settando il bit più significativo del byte alto del puntatore della matrice video, ovvero aggiungendo 128 al suo codice di schermo. Questo abilita l'inversione dei dati che lo compongono da parte del TED chip, e la sua conseguente visualizzazione. Nel caso venga richiesto un set alternativo di 256 caratteri, il bit 7 può essere settato, disabilitando così la possibilità d'inversione e permettendo la definizione dei nuovi caratteri.

Registro #8: Allacciamento della tastiera

Scrivendo in questo registro si provoca una scansione della matrice della tastiera ed un conseguente aggancio del dato richiesto. Leggendo questo registro, si ottiene in uscita il valore precedentemente agganciato.

Registro #9: Stato dell'interrupt

Esso rappresenta il registro di sorgente degli interrupts. Ogni interrupt viene registrato attraverso l'azzeramento di un particolare bit di questo

registro. Le possibili sorgenti sono:

Bit 1 — Interrupt di scansione (raster interrupt)
Bit 2 — Penna ottica (successiva espansione)
Bit 3 — Interrupt del timer #1
Bit 4 — Interrupt del timer #2
Bit 6 — Interrupt del timer #3
Bit 7 — Interrupt richiesto

Ogni singolo bit può essere resettato riponendolo a uno.

Registro #10: Maschera dell'interrupt

Esso rappresenta la maschera per il registro di stato dell'interrupt. Settando uno dei suoi bits si ottiene, da parte del corrispondente bit nel registro di stato, la segnalazione di un futuro interrupt. Il bit 0 rappresenta il bit più significativo del registro di comparazione della scansione dell'immagine, e pertanto non è incluso nella maschera (vedere registro #11 per la descrizione).

Registro #11: Comparazione della scansione dell'immagine

In un sistema televisivo NTSC, l'immagine viene composta da 262 linee di scansione (da 0 a 261), mentre in un sistema PAL le linee sono 312 (da 0 a 311). Per tenere conto di tutte queste linee, si è reso necessario un registro di nove bits. Il registro #11 comprende gli otto bits meno significativi, mentre il nono bit più significativo corrisponde al bit 0 del registro 10 di maschera dell'interrupt. Questo registro rappresenta una sorgente di interrupt. Quando il contatore delle linee raggiunge il valore in esso immagazzinato, viene generato un interrupt. Questa tecnica può essere impiegata per operazioni di frazionamento dello schermo. Dato che la sua esecuzione potrebbe causare un leggero ma rimarcabile ritardo, esso viene generato otto cicli prima della finestra dei caratteri, minimizzando in tal modo il lampeggio dello schermo. Nella configurazione a 25 linee di testo, le linee di scansione visibili sono quelle dalla 4 alla 203.

Registro #12: Posizione del cursore (Registro Alto)

Esso contiene i due bits più significativi del registro di posizione del cursore. Il bit 0 ed il bit 1 rappresentano rispettivamente i bits 8 e 9 della posizione del cursore.

Registro #13: Posizione del cursore (Registro Basso)

Gli otto bits meno significativi del registro di posizione del cursore sono

qui contenuti. Tale registro è formato da dieci bits, rappresentanti 1024 diverse locazioni di cursore.

Registro #14: Frequenza voce #1 (Registro Basso)

Questo registro contiene il byte basso della frequenza di base della voce #1. Tale voce può disporre solamente dell'oscillatore ad onda quadra come sorgente di suono.

Registro #15: Frequenza voce #2 (Registro Basso)

Questo registro contiene gli otto bits meno significativi della frequenza base della voce #2. Tale voce dispone tanto dell'oscillatore ad onda quadra quanto di quello a rumore bianco, selezionabili attraverso un bit contenuto nel registro #17.

Registro #16: Frequenza voce #2 (Registro Alto)

I bits 0 e 1 di questo registro rappresentano rispettivamente i due bits più significativi della frequenza base della voce #2.

Registro #17: Controllo del suono

I bits 0—3 di questo registro rappresentano il livello del volume di uscita (minimo=0 e massimo=8). Quando settati, il bit 4 abilita la voce #1, il bit 5 la voce #2 con l'oscillatore ad onda quadra ed il bit 6 la voce #2 con l'oscillatore a rumore bianco. Il bit 7 costituisce infine un bit di test.

Registro #18: Base dell'alta risoluzione

Registro a funzione multipla. I suoi bits 0 e 1 rappresentano i due bits più significativi della frequenza base della voce #1.

Il bit 2 viene utilizzato per indicare da quale memoria TED chip deve prelevare i dati dei caratteri. Posto a uno, seleziona la ROM, mentre se è azzerato sceglie la RAM.

I bits 3—5 vengono impiegati per determinare la posizione della base dell'alta risoluzione. Durante il prelevamento dei punti da parte del TED chip, i tre bits più significativi delle linee d'indirizzo, A13—A15, vengono copiati dal bit 5 al bit 3.

Registro #19: Base dei caratteri

Il bit 0 di questo registro è un bit segnalatore a sola lettura descrivente lo stato dei due registri fantasma #62 e #63. Se posto a uno, il TED chip opera a partire dalla memoria ROM. Nel caso venga azzerato, non è possibile accedere ai registri del TED chip.

Il bit 1 provvede a forzare il modo a singolo clock quando settato, inibendo la doppia velocità del clock mentre lo schermo è disabilitato.

I bits 2—7 comprendono la base dei dati dei caratteri. Questi sei bits assegnano 64 diverse zone ad incrementi di 1 Kbyte per i dati dei caratteri. Per cambiare set di caratteri, il registro di base deve essere assegnato al valore appropriato in funzione dell'allocazione del nuovo set, quindi il bit di selezione RAM/ROM (bit 2 del registro #18) deve essere azzerato. Il TED chip si riferirà adesso alla zona di RAM designata per prelevare l'informazione relativa ai nuovi caratteri.

Registro #20: Base della matrice video

I cinque bits più significativi di questo registro (bits 3—7) controllano la base della matrice video. Essi determinano quale blocco di 2 Kbytes di memoria deve rappresentare i puntatori della matrice video ed i dati di attribuzione (memoria colore e di schermo). Attraverso un accorto utilizzo del registro di comparazione della scansione dell'immagine, è possibile definire uno schermo frazionato comprendente due differenti sets di dati di caratteri e colori provenienti da due distinte aree di memoria.

Registro #21: Colore dello sfondo

Questo registro comprende quattro bits di colore e tre bits di luminosità. In tal modo sono disponibili otto diversi gradi di luminosità per ciascuno dei sedici colori implementati. I bits 0—3 definiscono il colore dello sfondo, mentre i bits 4—6 ne determinano il grado di luminosità.

Registro #22: Colore dei caratteri

I bits 0—3 definiscono il colore dei caratteri, mentre i bits 4—6 ne determinano il grado di luminosità.

Registro #23: Multicolor #1

I bits 0—3 definiscono il colore multicolor #1, utilizzabile unicamente nel modo del colore esteso, mentre i bits 4—6 ne determinano il grado di luminosità.

Registro #24: Multicolor #2

I bits 0—3 definiscono il colore multicolor #2, anch'esso utilizzabile soltanto nel modo del colore esteso, mentre i bits 4—6 ne determinano il grado di luminosità.

Registro #25: Colore della cornice

I bits 0—3 definiscono il colore della cornice, mentre i bits 4—6 ne determinano il grado di luminosità.

Registro #26: Posizione del carattere (Registro Alto)

I bits 0 e 1 rappresentano i due bits più significativi del registro di posizione del carattere. Tale registro viene utilizzato dal TED chip per contare la linea di schermo sulla quale visualizzare i caratteri. Ogni volta che una linea di schermo, formata da otto linee di scansione dell'immagine, è stata visualizzata, questo registro viene incrementato di 40 unità.

Registro #27: Posizione del carattere (Registro Basso)

Esso contiene gli otto bits meno significativi del registro di posizione del carattere.

Registro #28: Contatore di scansione (Registro Alto)

Il bit 0 rappresenta il bit più significativo del registro di linea verticale formato da nove bits. Questo registro viene utilizzato dal TED chip per contare la corrente linea di scansione dell'immagine visualizzata. Esso varia da 0 a 261 per il sistema NTSC, e da 0 a 311 per il sistema PAL.

Registro #29: Contatore di scansione (Registro Basso)

Esso contiene gli otto bits meno significativi del registro contatore di scansione dell'immagine.

Registro #30: Posizione orizzontale

Esso comprende gli otto bits più significativi del registro di posizione orizzontale formato da nove bits. Il suo bit meno significativo non è disponibile in quanto cambia troppo velocemente stato per poter essere convenientemente utilizzato. Questo registro viene incrementato da 0 a 455, ma dato che solo i suoi otto bits più significativi vengono messi a disposizione, esso varia in realtà da 0 a 288. Visto che tale registro viene

molto rapidamente aggiornato, risulta estremamente appropriato per generare numeri casuali.

Registro #31: Lampeggio

I bits 0—3 comprendono il registro del rapporto di lampeggio, il quale contiene il corrente valore dell'omonimo timer. Tale registro viene incrementato una volta per ogni schermo. Quando raggiunge il massimo valore consentito, viene generato un segnale di 2 Hz allo scopo di inizializzare il video in negativo ed ogni carattere lampeggiante.

I bits 4—6 rappresentano il registro del subindirizzo verticale, il quale provvede a contare le otto linee di scansione dell'immagine per ogni linea dello schermo.

Registri #62 e #63

Questi due registri non fanno realmente parte del TED chip, ma vengono invece utilizzati per controllare la configurazione di memoria del sistema. Scrivendo nel registro #62 si provvede a selezionare la ROM nel banco $8000—$FFFF, escludendo contemporaneamente le zone relative a procedure di I/O ed il TED da $FD00 a $FF3F. Scrivendo infine nel registro #63, si seleziona la RAM nello stesso intervallo di memoria, pertanto il BASIC può essere disabilitato.

Notate che tutti i registri del TED chip sono attivi tanto in lettura quanto in scrittura, per cui è necessario prestare molta attenzione nel modificare i registri dal #26 al #31, i quali costituiscono delle forme di controllo interno. Assegnargli particolari valori potrebbe provocare un tremolio dello schermo non desiderato.

APPENDICE 7

Caratteristiche di un buon Assemblatore

Prima o poi sentirete senza dubbio la necessità di passare ad utilizzare un completo assemblatore, con caratteristiche decisamente superiori a quelle riscontrabili in TEDMON. Vi renderete rapidamente conto di come l'impiego di TEDMON per assemblare programmi di media o ampia grandezza risulti notevolmente tedioso e confuso. Ad esempio, considerate un salto condizionato in avanti riferito ad una locazione sensibilmente distante:

```
2000 BEQ $2008
```

Per ottenere l'indirizzo di destinazione di questo salto, dovete calcolare il numero di locazioni interposte fra il salto e la sua destinazione, e quindi addizionare tale valore al suo indirizzo originale. Questo esempio illustra una delle varie limitazioni proprie di un assemblatore semplice. Consideriamo adesso il seguente programma, che provvede a stampare nella linea superiore dello schermo i caratteri alfabetici dalla A alla Z (codici 1—26).

```
2000 LDA #$01
2002 STA $03
2004 LDY #$00
2006 LDA $03
2008 STA $0C00,Y
200B INC $03
200D INY
200E CPY #$1A
2010 BNE $2006
2012 BRK
```

Sebbene questo programma sia estremamente corto, è comunque abbastanza difficile da seguire senza adeguati commenti ausiliari. Lo stesso programma scritto per mezzo di un assemblatore completo apparirà sotto una forma simile a quella presentata di seguito:

```
10          ORG   $2000      ; inizio programma in $2000
15 VIDEO    =     $0C00      ; indirizzo di base dello schermo
20 CARAT    =     $03          byte immagazzinamento caratteri
25
30   registro Y usato come indice per lo schermo e come
```

```
35 ; contatore fino a 26 (numero dei caratteri)
40 ;
45          LDA  #$01     ; carica il codice del carattere A
50          STA  CARAT    ; lo scarica nella locazione CARAT
55          LDY  #$00     ; inizializza il nostro contatore
60 RIPETI   LDA  CARAT    ; carica il corrente carattere
65          STA  VIDEO,Y  ; lo scarica nello schermo
70          INC  CARAT    ; aggiorna il codice del carattere
75          INY           ; incrementa il contatore
80          CPY  #$1A     ; compara il contatore con 26
85          BNE  RIPETI   ; se diverso mostra il prossimo
90          BRK             uscita dal programma
```

Come avrete notato, un assemblatore completo è strutturato in modo da facilitare il lavoro al programmatore, ma non al computer. L'esempio precedente potrebbe forse sembrarvi una banalità sovracommentata, ma permette d'illustrarvi il tipo di documentazione implementabile all'interno di un programma.
Disponendo di un assemblatore completo è possibile utilizzare etichette al posto di indirizzi assoluti, evitando così al programmatore il disagio di doversi calcolare i salti relativi come nel caso di TEDMON. L'ordinamento delle istruzioni, importante tanto in fase di costruzione quanto di verifica, viene assicurato tramite l'impiego di normali numeri di linea.
Lo strano mnemonico ORG presente nella linea 10 è conosciuto come uno pseudo—codice. Gli assemblatori necessitano di particolari informazioni addizionali, come ad esempio dove assemblare il codice sorgente. La linea 10 si limita appunto ad assegnare l'inizio dell'assemblaggio vero e proprio all'indirizzo $2000.
La maggior parte degli assemblatori permette d'immagazzinare il codice sorgente (ovvero non ancora propriamente assemblato) su nastro o disco, di stamparlo e modificarlo a vostro piacimento, aggiungendo o cancellando linee di programma. A seguire vi presentiamo le principali caratteristiche da considerare nella scelta di un assemblatore adatto alle vostre esigenze.

Etichette

Quasi tutti gli assemblatori contemplano l'uso simultaneo di etichette convenzionali ed indirizzi risultanti, da impiegare come parametri all'interno delle istruzioni. Questa possibilità è da tenere senza dubbio in estrema considerazione in quanto alleggerisce sensibilmente la necessità di calcolare gli indirizzi di destinazione, provocando un considerevole risparmio di tempo nella costruzione dei programmi.
Esistono fondamentalmente due tipi di etichette impiegabili all'interno di una sorgente assembly:

[1] INTERNE: si riferiscono ad una locazione interna al programma assemblato. Ad esempio:

```
 5          ORG $2000
10          JMP FUORI
 .
 .
 .
30 FUORI    RTS
```

In questo caso, l'etichetta FUORI viene denominata interna in quanto la locazione che identifica risiede all'interno del programma assemblato.

[2] ESTERNE: si riferiscono ad una locazione esterna al programma assemblato. Ad esempio:

```
 5          ORG $2000
10 SET      =2
 .
 .
 .
30          JSR SET
```

L'etichetta SET viene denominata esterna in quanto la locazione che identifica risiede all'esterno del programma assemblato.

Messaggi di errore

Altro aspetto importante da non sottovalutare in un assemblatore è rappresentato dalla capacità comunicativa dei suoi messaggi di errore. Niente urta più di un assemblatore che segnala una condizione di errore senza fornire delucidazioni circa la sua origine. Fortunatamente questa categoria di assemblatori sembra ormai estinta. Infatti è importante notare come quasi tutti i modelli presenti sul mercato provvedano a visualizzare un esauriente messaggio oppure, nella peggiore delle ipotesi, un numero di codice che rimanda alla consultazione del manuale fornito in dotazione. Tenete sempre conto di questo all'atto di acquistare un assemblatore, in quanto la procedura di verifica richiede generalmente un considerevole quantità di tempo nello sviluppo di un programma.

Direttive dell'assemblatore

Questa sezione illustra una serie d'istruzioni che possono risultare più o meno importanti nell'assemblaggio dei vostri programmi. Le direttive

dell'assemblatore sono dei particolari comandi addizionali che facilitano la formattazione dei vostri listati, la gestione della memoria, e più generalmente contribuiscono a rendere scorrevole l'opera dell'assemblatore. Sebbene i nomi di questi comandi possano variare da assemblatore ad assemblatore, vi presentiamo nel seguito la descrizione di alcuni di essi.

[1] INIZIO DELL'ASSEMBLAGGIO: tutti gli assemblatori dispongono di un'istruzione che permette di allocare l'indirizzo iniziale di assemblaggio del codice sorgente. Potete dare per scontato che, in una forma o nell'altra, questa possibilità è presente in qualsiasi assemblatore.

[2] RISERVA DI MEMORIA: questo comando viene utilizzato per riservare zone di memoria da impiegare nel vostro programma. Esso vi permette di creare aree vuote all'interno delle quali possono successivamente essere immagazzinati dei dati di qualsiasi natura. La maggior parte degli assemblatori dispone di questa possibilità.

[3] IMMAGAZZINAMENTO DATI: è importante che un buon assemblatore permetta un facile immagazzinamento di dati alfanumerici a partire da una desiderata locazione di memoria. Un tale comando deve essere in grado di memorizzare tanto valori numerici quanto stringhe di caratteri ASCII.

[4] FORMATO DI STAMPA: questo comando provvede generalmente a visualizzare una stampa finale del codice sorgente in forma tabulata, ovvero allineando i vari campi secondo il seguente ordine:

<A> Etichetta
<B> Codice mnemonico
<C> Operandi
<D> Eventuali commenti

Il principale vantaggio di questo comando è rappresentato da una lettura notevolmente facilitata del listato sorgente. Questa possibilità non si può qualificare come indispensabile, ma tuttavia costituisce un deciso elemento preferenziale nella scelta di un assemblatore.

[5] BASI NUMERICHE CONTEMPLATE: un buon assemblatore deve accettare numeri espressi nelle seguenti basi:

<*> Decimale (base 10)
<*> Esadecimale (base 16)
<*> Binaria (base 2)
<*> Ottale (base 8)

Il sistema di numerazione ottale, oggigiorno in declino, non è comunque da considerarsi indispensabile come gli altri tre. La base 10 è decisamente la più vicina all'uomo, in quanto viene utilizzata regolarmente nella vita di tutti i giorni. L'esadecimale risulta invece estremamente pratico nel trattamento degli indirizzi di memoria. Il sistema binario è infine il più indicato nelle procedure di manipolazione dei singoli bits, come ad esempio i mascheramenti. Come regola generale, un assemblatore deve adottare la seguente nomenclatura:

<*> Numeri decimali senza prefisso
<*> Numeri esadecimali preceduti dal simbolo $
<*> Numeri binari preceduti dal simbolo %
<*> Numeri ottali preceduti dal simbolo @

Costruzione di programmi voluminosi

Utilizzando un assemblatore, il codice sorgente occupa generalmente una quantità di memoria notevolmente superiore a quella del codice macchina generato. Vediamo un piccolo esempio:

```
LDA #$08
STA $200A
```

Questo programmino occupa cinque bytes effettivi, mentre la sua sorgente richiede probabilmente dai venti ai trenta bytes. Il C16 dispone di sedici Kbytes di memoria utilizzabile per contenere i vostri programmi in linguaggio macchina. La generazione di un tale ammontare di codice macchina richiederebbe fra sessanta e cento Kbytes di codice sorgente, in funzione del formato di memorizzazione adottato dall'assemblatore. È perfettamente ovvio che il computer non è in grado di trattare un tale volume d'informazione in una sola volta. Il metodo adottato per risolvere questo problema consiste appunto nel concatenare vari files sorgenti, ciascuno di essi assemblato separatamente, i quali contribuiscono a generare un unico programma voluminoso. Se pensate di voler scrivere soltanto piccoli programmi, allora questa tecnica non risulta necessaria.
Giunti a questo punto, vale la pena segnalare i due metodi principali adottati da un assemblatore per generare il codice macchina risultante del vostro programma.

[1] L'assemblatore immagazzina direttamente in memoria il codice macchina generato. Questa tecnica risulta estremamente limitativa, in quanto il ridotto ammontare della memoria del C16 permette di costruire soltanto programmi piuttosto piccoli. Non dovete dimenticare che un assembla-

tore, che abbia a disposizione la maggior parte delle caratteristiche precedentemente illustrate, necessita probabilmente di almeno dieci Kbytes per poter operare convenientemente.

[2] L'assemblatore consente la registrazione su nastro o disco del codice macchina generato, lasciando in tal modo a completa disposizione della sorgente tutti i sedici Kbytes utilizzabili. Questa tecnica è preferibile per costruire grossi programmi, in quanto elimina ogni conflitto fra programma ed assemblatore.

Esistono altre variazioni su questi due temi principali, ognuna con i suoi pregi ed i suoi difetti. Il metodo utilizzato per immagazzinare il codice macchina risultante deve essere tenuto in seria considerazione al momento di acquistare un assemblatore per un computer con forti limitazioni di memoria quale il C16.

Macroistruzioni

Sono sempre più numerosi gli assemblatori che comprendono un'interessante possibilità di utilizzare le cosiddette "macroistruzioni". Una macro rappresenta una predefinita serie d'istruzioni identificata per mezzo di un'etichetta. Dopo che una macro è stata definita, successivi riferimenti al suo nome identificatore inseriscono automaticamente il codice associato a partire dalla sua definizione. Ad una macro possono essere inviati anche dei parametri. Vediamone un esempio:

```
10 MACRO INCBYTE    INC ?1
15                  BNE ?3
20                  INC ?2
25 ?3               NOP
30 MACRO END
```

A questo punto, la nostra macro è stata definita. Se adesso consideriamo la linea

```
50                  INCBYTE $05,$06
```

il seguente codice viene automaticamente inserito:

```
                    INC $05
                    BNE L01
                    INC $06
          L01       NOP
```

Tale codice corrisponde alla definizione della macro, con i parametri (?1, ?2, ?3) assegnati nella linea 50.
Le macro risultano estremamente pratiche e maneggevoli nei casi in cui parti di codice vengono ricorsivamente utilizzate con diversi parametri.

Funzioni matematiche

La maggior parte degli assemblatori comprendono l'impiego di semplici funzioni matematiche, quali addizioni, sottrazioni, moltiplicazioni e divisioni. Tali funzioni alleggeriscono sensibilmente il programmatore della necessità di dover effettuare mentalmente i propri calcoli. Vediamo un piccolo esempio applicativo:

```
10 BASE    = $03
15 BASE2   = $04
20 BASE3   = $06
25         INC BASE
30         INC BASE2
35         INC BASE3
```

Disponendo di un assemblatore comprendente funzioni matematiche, il codice sorgente qui sopra può essere digitato nella forma seguente:

```
10 BASE    = $03
15         INC BASE
20         INC BASE+1
25         INC BASE*2
```

Questa possibilità non deve essere considerata come indispensabile, ma come un'interessantissima e pratica caratteristica preferenziale.

Opzione di biblioteca

Alcuni assemblatori permettono d'immagazzinare su nastro o disco porzioni di codice sorgente, le quali possono in seguito essere richiamate ed inserite in qualsiasi altro programma. Se, ad esempio, avete scritto una routine per gestire l'input attraverso un joystick, potete conservarla nella vostra biblioteca ed inserirla quindi in tutti quei programmi che necessitano di una tale routine.

Funzioni di trattamento testo

Gli assemblatori più recenti iniziano a disporre di caratteristiche tradizionalmente appartenenti a programmi di word processing (trattamento te-

sti). Alcuni di essi vi permettono di spostare blocchi di codice sorgente da un posto ad un altro, ricercare e sostituire particolari istruzioni, e numerose altre funzioni che facilitano sensibilmente la programmazione. Anche quest'ultima rappresenta una delle tante possibilità supplementari, forse non indispensabili, ma certamente molto utili per velocizzare la costruzione del codice sorgente.

Opzione di sfasatura (offset)

Questa opzione provvede ad immagazzinare il codice assemblato a partire da un indirizzo diverso da quello corrispondente alla sua posizione esecutiva. Essa permette di caricare il codice in una particolare zona e quindi averlo automaticamente trasferito al suo proprio indirizzo operativo quando richiesto. Tale opzione risulta inoltre molto utile nella programmazione delle EPROMS (chips di memoria a sola lettura programmabili e cancellabili).

In questa sezione abbiamo trattato alcune delle più comuni caratteristiche implementabili in un assemblatore. Se dedicate un poco del vostro tempo ad esaminare i principali assemblatori presenti sul mercato, noterete senza dubbio la presenza di ulteriori possibilità che non abbiamo avuto modo d'illustrare. Infatti, ci siamo limitati in questa appendice a considerare unicamente quelle che ci sono sembrate le più utili ed interessanti. La scelta sull'assemblatore da acquistare è pertanto lasciata alla vostra personale preferenza.

Codici ASCII

Questa tabella vi mostra tutti i possibili caratteri ottenibili con PRINT CHR$(X) per X che varia da 0 a 255. Inversamente, essa vi mostra anche valori ottenibili con PRINT ASC("X") con X rappresentante un qualsiasi carattere digitabile. Il suo impiego risulta estremamente utile per identificare il carattere ricevuto attraverso un comando GET, per passare dal set maiuscolo al set minuscolo e viceversa, e per stampare speciali caratteri di controllo (tipo la disabilitazione del tasto SHIFT) non inseribili fra gli apici.

STAMPA	CHR$	STAMPA	CHR$	STAMPA	CHR$	STAMPA	CHR$
	0	↓	17		34	3	51
	1	RVS ON	18		35	4	52
	2	CLR/HOME	19	$	36	5	53
STOP	3	INST DEL	20	%	37	6	54
	4		21	&	38	7	55
BIANCO	5		22	'	39	8	56
	6		23	(	40	9	57
	7		24	)	41		58
DISAB.SHIFT C=	8		25	•	42		59
ABILITA' SHIFT C=	9		26	+	43	<	60
	10	ESCAPE	27		44	=	61
	11	ROSSO	28		45	>	62
	12	–	29		46	?	63
RETURN	13	VERDE	30	/	47	@	64
MINUSCOLE	14	BLU	31	0	48	A	65
	15	SPAZIO	32	1	49	B	66
	16	!	33	2	50	C	67

STAMPA	CHR$	STAMPA	CHR$	STAMPA	CHR$	STAMPA	CHR$
D	68		97		126	VERDE CH.	155
E	69		98		127	PORPORA	156
F	70		99		128	←	157
G	71		100	ARANCIO	129	GIALLO	158
H	72		101	FLASH ON	130	CIANO	159
I	73		102	SHIFT RUN/STOP	131	SPACE	160
J	74		103	FLASH OFF	132		161
K	75		104	f1	133		162
L	76		105	f3	134		163
M	77		106	f5	135		164
N	78		107	f7	136		165
O	79		108	f2	137		166
P	80		109	f4	138		167
Q	81		110	f6	139		168
R	82		111	f8	140		169
S	83		112	SHIFT RETURN	141		170
T	84		113	MAIUSCOLE	142		171
U	85		114		143		172
V	86		115	NERO	144		173
W	87		116	↑	145		174
X	88		117	RVS OFF	146		175
Y	89		118	CLEAR HOME	147		176
Z	90		119	INST DEL	148		177
[	91		120	MARRONE	149		178
£	92		121	GIALLO/VERDE	150		179
]	93		122	ROSA	151		180
↑	94		123	BLU/VERDE	152		181
←	95		124	BLU CH.	153		182
	96		125	BLU	154		183

STAMPA	CHR$	STAMPA	CHR$	STAMPA	CHR$	STAMPA	CHR$
	184		186		188		190
	185		187		189		191

CODICI	192-223	COME	96-127
CODICI	224-254	COME	160-190
CODICE	255	COME	126

APPENDICE 9

Codici di schermo

I codici di schermo elencati qui sotto corrispondono ai valori da immagazzinare nelle appropriate locazioni di schermo per visualizzare il corrispondente carattere. Sono disponibili due sets, ma non simultaneamente. Per passare da uno all'altro set, è necessario premere il tasto COMMODORE mentre si tiene premuto il tasto SHIFT.

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	T	t	20	(		40
A	a	1	U	u	21	)		41
B	b	2	V	v	22	•		42
C	c	3	W	w	23	+		43
D	d	4	X	x	24			44
E	e	5	Y	y	25			45
F	f	6	Z	z	26			46
G	g	7	[		27	/		47
H	h	8	£		28	0		48
I		9	]		29	1		49
J	i	10	↑		30	2		50
K	k	11	←		31	3		51
L	l	12	SPAZIO		32	4		52
M	m	13	!		33	5		53
N	n	14			34	6		54
O	o	15			35	7		55
P	p	16	$		36	8		56
Q	q	17	%		37	9		57
R	r	18	&		38			58
S	s	19			39			59

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
<		60		T	84			108
=		61		U	85			109
>		62		V	86			110
?		63		W	87			111
		64		X	88			112
	A	65		Y	89			113
	B	66		Z	90			114
	C	67			91			115
	D	68			92			116
	E	69			93			117
	F	70			94			118
	G	71			95			119
	H	72	SPAZIO		96			120
	I	73			97			121
	J	74			98		☑	122
	K	75			99			123
	L	76			100			124
	M	77			101			125
	N	78			102			126
	O	79			103			127
	P	80			104			
	Q	81			105			
	R	82			106			
	S	83			107			

I codici da 128 a 255 sono le immagini invertite dei codici da 0 a 127

Cassetta in dotazione

La cassetta fornita in dotazione contiene tre programmi dimostrativi in linguaggio macchina svolgenti altrettante utili funzioni di aiuto alla programmazione. Per la vostra comodità, tali programmi vengono caricati attraverso un listato BASIC, secondo la tecnica descritta nel capitolo numero 2. Questo per evitarvi il disagio di doverli caricare per mezzo di TEDMON, e soprattutto allo scopo di permettervi di controllare il loro avvenuto caricamento attraverso il comando LIST.

Append

Questa routine provvede ad unire un vostro programma BASIC in coda ad un'altro già memorizzato nel computer. È importante verificare che il minore numero di linea del secondo programma sia superiore al maggiore numero di linea del primo programma. Per utilizzarla, effettuate nell'ordine le seguenti operazioni:

<A> Caricate la routine APPEND in memoria
<B> Digitate RUN e premete RETURN
<C> Caricate il primo programma BASIC
<D> Digitate SYS 16331 e premete RETURN
<E> Caricate il secondo programma BASIC
<F> Digitate SYS 16355 e premete RETURN

```
10 POKE55,202:POKE56,63:CLR
20 FORI=16331+34:READA:POKEI,A:C=C+A:NEXT
30 IFC<>4094THENPRINT"ERRORE NELLE ISTRUZIONI
   DATA":STOP
40 DATA165,43,141,238,63,165,44,141,239,63,56,165
   ,45,233,2,133,43,165
50 DATA46,233,0,133,44,96,173,238,63,13,3,43
   ,173,239,63,133,44,96
60 NEW
```

```
3FCB  A5  2B      LDA  $2B
3FCD  8D  EE  3F  STA  $3FEE
3FD0  A5  2C      LDA  $2C
```

```
3FD2  8D  EF  3F   STA  $3FEF
3FD5  38           SEC
3FD6  A5  2D       LDA  $2D
3FD8  E9  02       SBC  #$02
3FDA  85  2B       STA  $2B
3FDC  A5  2E       LDA  $2E
3FDE  E9  00       SBC  #$00
3FE0  85  2C       STA  $2C
3FE2  60           RTS
3FE3  AD  EE  3F   LFA  $3FEE
3FE6  85  2B       STA  $2B
3FE8  AD  EF  3F   LDA  $3FEF
3FEB  85  2C       STA  $2C
3FED  60           RTS
```

Restore

Questa routine permette di recuperare un programma BASIC accidentalmente cancellato con NEW, DELETE, oppure premendo il tasto RESET. Essa richiede almeno 107 bytes liberi in fondo alla memoria per poter funzionare correttamente. È importante precisare che un programma non può essere recuperato se dopo la sua perdita sono state digitate nuove linee. Notate che il primo numero di linea, andato distrutto nella cancellazione, viene automaticamente posto a zero in modo da evitare ogni conflitto con i successivi. Per rendere funzionante questa routine, è necessario effettuare le seguenti operazioni preliminari:

<A> Caricate la routine RESTORE in memoria
<B> Digitate RUN e premete RETURN
<C> Digitate MONITOR e premete RETURN
<D> Salvate una copia–lavoro della routine con:
S "COPIA–LAVORO",01,3F8E,3FF9 (su nastro)
S "COPIA–LAVORO",08,3F8E,3FF9 (su disco)
<E> Ritornate al BASIC digitando X e premendo RETURN

A questo punto, supponendo di voler recuperare un programma erroneamente cancellato, eseguite quanto segue:

<A> Caricate in memoria il programma COPIA–LAVORO con:
LOAD "COPIA–LAVORO" ,1,1 (da nastro)
LOAD "COPIA–LAVORO" ,8,1 (da disco)
<B> Digitate NEW e premete RETURN

<C> Digitate SYS 16270 e premete RETURN
<D> Digitate CLR e premete RETURN

Il programma precedentemente cancellato è adesso nuovamente disponibile. Ricordatevi sempre di salvarlo prima di eseguirlo o modificarlo ulteriormente.

```
 10 POKE55,141:POKE56,63:CLR
 20 FORI=16270TO16376:READA:C=C+A:POKEI,A:NEXT
 30 IFC<>11190THENPRINT"ERRORE NELLE ISTRUZIONI
    DATA":END
 40 DATA165,43,133,3,165,44,133,4,160,4,177,3,240,3,200,208
 50 DATA249,200,152,32,237,63,160,0,165,3,145,43,200,165,4,145
 60 DATA43,200,169,0,145,43,200,145,43,1,65,43,133,3,165,44,133
 70 DATA4,160,0,169,4,32,237,63,177,3,24,0,7,32,235,63,169
 80 DATA0,240,245,32,235,63,177,3,240,4,169,0,240,234,32,235
 90 DATA63,32,235,63,165,3,133,45,165,4,133,46,96,169,1,24
100 DATA101,3,133,3,169,0,101,4,133,4,96
```

```
3F8E  A5 2B     LDA  $2B
3F90  85 03     STA  $03
3F92  A5 2C     LDA  $2C
3F94  85 04     STA  $04
3F96  A0 04     LDY  #$04
3F98  B1 03     LDA  ($03),Y
3F9A  F0 03     BEQ  $3F9F
3F9C  C8        INY
3F9D  D0 F9     BNE  $3F98
3F9F  C8        INY
3FA0  98        TYA
3FA1  20 ED 3F  JSR  $3FED
3FA4  A0 00     LDY  #$00
3FA6  A5 03     LDA  $03
3FA8  91 2B     STA  ($2B),Y
3FFA  C8        INY
3FAB  A5 04     LDA  ?04
3FAD  91 2B     STA  ($2B),Y
3FAF  C8        INY
3FB0  A9 00     LDA  #$00
3FB2  91 2B     STA  ($2B),Y
3FB4  C8        INY
3FB5  91 2B     STA  ($2B),Y
3FB7  A5 2B     LDA  $2B
```

```
3FB9  85  03      STA  $03
3FBB  A5  2C      LDA  $2C
3FBD  85  04      STA  $04
3FBF  A0  00      LDY  #$00
3FC1  A9  04      LDA  #$04
3FC3  20  ED  3F  JSR  $3FED
3FC6  B1  03      LDA  ($03),Y
3FC8  F0  07      BEQ  $3FD1
3FCA  20  EB  3F  JSR  $3FEB
3FCD  A9  00      LDA  #$00
3FCF  F0  F5      BEQ  $3FC6
3FD1  20  EB  3F  JSR  $3FEB
3FD4  B1  03      LDA  ($03),Y
3FD6  F0  04      BEQ  $3FDC
3FD8  A9  00      LDA  #$00
3FDA  F0  EA      BEQ  $3FC6
3FDC  20  EB  3F  JSR  $3FEB
3FDF  20  EB  3F  JSR  $3FEB
3FE2  A5  03      LDA  $03
3FE4  85  2D      STA  $2D
3FE6  A5  04      LDA  $04
3FE8  85  2E      STA  $2E
3FEA  60          RTS
3FEB  A9  01      LDA  #$01
3FED  18          CLC
3FEE  65  03      ADC  $03
3FF0  85  03      STA  $03
3FF2  A9  00      LDA  #$00
3FF4  65  04      ADC  $04
3FF6  85  04      STA  $04
3FF8  60          RTS
```

Lista delle variabili

Questa utile routine fornisce un elenco di tutte le variabili inizializzate in quel momento all'interno di un programma BASIC, ciascuna accompagnata da una sigla indicante il tipo. Le sigle in questione sono tre:

FLO = virgola mobile
INT = intera
STR = stringa

Gli arrays vengono ignorati in quanto sono tutti generalmente definiti

nello stesso posto da un'istruzione DIM. Per utilizzare correttamente questa routine, eseguite le seguenti operazioni:

<A> Caricate la routine LISTA VARIABILI in memoria
<B> Digitate RUN e premete RETURN
<C> Caricate il vostro programma BASIC
<D> Digitate RUN e premete RETURN
<E> Premete RUN/STOP al momento desiderato
<F> Digitate SYS 16100 e premete RETURN

A questo punto apparirà sullo schermo una lista di tutte le variabili inizializzate dal programma prima che la sua esecuzione sia stata arrestata per mezzo del tasto RUN/STOP. La stampa può essere momentaneamente interrotta premendo lo spazio, quindi ripresa premendo il tasto C. Affinchè questa routine possa funzionare correttamente, è importante che il programma BASIC del quale si voglia elencare le variabili non effettui alcun assegnamento (POKE) nell'intervallo di memoria compreso fra le locazioni 16100 e 16384 (residenza della nostra routine).

```
 10 POKE55,227:POKE56,62:CLR
 20 FORI=16100TO16100+273:READA:POKEI,A:
    C=C+A:NEXT
 30 IFC <>30678THENPRINT"ERRORE NELLE IST
    RUZIONI DATA":STOP
 40 DATA165,45,133,2,165,46,133,3,165,2,197,47,208,6,165,3
 50 DATA197,48,240,40,160,0,177,2,141,22,7,63,201,128,176,30,200
 60 DATA177,2,201,128,176,65,32,106,63,3,2,211,63,169,7,24,101
 70 DATA2,133,2,169,0,101,3,133,3,76,236,62,96,56,233,128
 80 DATA141,227,63,200,177,2,201,128,240,24,56,233,128
    ,141,228,63
 90 DATA32,134,63,32,147,63,174,238,63,3,2,164,63,32,205,63,76
100 DATA13,63,169,32,76,49,63,240,24,56,233,128,141,228,63,32
110 DATA134,63,32,147,63,174,240,63,32,1,64,63,32,205,63,76,13
120 DATA63,169,32,76,80,63,201,0,240,19,141,228,63,32,134,63
130 DATA32,147,63,174,239,63,32,164,63,3,2,205,63,96,169,32,76
140 DATA110,63,173,227,63,32,183,63,173,228,63,32,183,63,96,162
150 DATA15,142,241,63,169,32,32,183,63,1,74,241,63,202
    ,208,242,96
160 DATA160,3,142,242,63,189,229,63,32,1,83,63,174,242
    ,63,232,136
170 DATA208,240,96,141,243,63,140,245,63,142,244,63,32
    ,210,255,174
180 DATA244,63,172,245,63,173,243,63,96,169,13,32,183,63,96,32
```

```
190 DATA228,255,201,32,240,1,96,32,228,2,55,201,67,208,249,96,0
200 DATA0,73,78,84,70,76,79,83,84,82,0,3,6,0,0,0,0,0
```

```
3EE4   A5  2D       LDA  $2D
3EE6   85  02       STA  $02
3EE8   A5  2E       LDA  $2E
3EEA   85  03       STA  $03
3EEC   A5  02       LDA  $02
3EEE   C5  2F       CMP  $2F
3EF0   D0  06       BNE  $3EF8
3EF2   A5  03       LDA  $03
3EF4   C5  30       CMP  $30
3EF6   F0  28       BEQ  $3F20
3EF8   A0  00       LDY  #$00
3EFA   B1  02       LDA  ($02),Y
3EFC   8D  E3  3F   STA  $3FE3
3EFF   C9  80       CMP  #$80
3F01   BO  1E       BCS  $3F21
3F03   C8           INY
3F04   B1  02       LDA  ($02),Y
3F06   C9  80       CMP  #$80
3F08   B0  41       BCS  $3F4B
3F0A   20  6A  3F   JSR  $3F6A
3F0D   20  D3  3F   JSR  $3FD3
3F10   A9  07       LDA  #$07
3F12   18           CLC
3F13   65  02       ADC  $02
3F15   85  02       STA  $02
3F17   A9  00       LDA  #$00
3F19   65  03       ADC  $03
3F1B   85  03       STA  $03
3F1D   4C  EC  3E   JMP  $3EEC
3F20   60           RTS
3F21   38           SEC
3F22   E9  80       SBC  #$80
3F24   8D  E3  3F   STA  $3FE3
3F27   C8           INY
3F28   B1  02       LDA  ($02),Y
3F2A   C9  80       CMP  #$80
3F2C   F0  18       BEQ  $3F46
3F2E   38           SEC
3F2F   E9  80       SBC  #$80
3F31   8D  E4  3F   STA  $3FE4
```

```
3F34  20  86  3F  JSR  $3F86
3F37  20  93  3F  JSR  $3F93
3F3A  AE  EE  3F  LDX  $3FEE
3F3D  20  A4  3F  JSR  $3FA4
3F40  20  CD  3F  JSR  $3FCD
3F43  4C  0D  3F  JMP  $3F0D
3F46  A9  20      LDA  #$20
3F48  4C  31  3F  JMP  $3F31
3F4B  F0  18      BEQ  $3F65
3F4D  38          SEC
3F4E  E9  80      SBC  #$80
3F50  8D  E4  3F  STA  $3FE4
3F53  20  86  3F  JSR  $3F86
3F56  20  93  3F  JSR  $3F93
3F59  AE  F0  3F  LDX  $3FF0
3F5C  20  A4  3F  JSR  $3FA4
3F5F  20  CD  3F  JSR  $3FCD
3F62  4C  0D  3F  JMP  $3F0D
3F65  A9  20      LDA  #$20
3F67  4C  50  3F  JMP  $3F50
3F6A  C9  00      CMP  #$00
3F6C  F0  13      BEQ  $3F81
3F6E  8D  E4  3F  STA  $3FE4
3F71  20  86  3F  JSR  $3F86
3F74  20  93  3F  JSR  $3F93
3F77  AE  EF  3F  LDX  $3FEF
3F7A  20  A4  3F  JSR  $3FA4
3F7D  20  CD  3F  JSR  $3FCD
3F80  60          RTS
3F81  A9  20      LDA  #$20
3F83  4C  6E  3F  JMP  $3F6E
3F86  AD  E3  3F  LDA  $3FE3
3F89  20  B7  3F  JSR  $3FB7
3F8C  AD  E4  3F  LDA  $3FE4
3F8F  20  B7  3F  JSR  $3FB7
3F92  60          RTS
3F93  A2  0F      LDX  #$0F
3F95  8E  F1  3F  STX  $3FF1
3F98  A9  20      LDA  #$20
3F9A  20  B7  3F  JSR  $3FB7
3F9D  AE  F1  3F  LDX  $3FF1
3FA0  CA          DEX
3FA1  D0  F2      BNE  $3F95
3FA3  60          RTS
```

```
3FA4  A0  03      LDY  #$03
3FA6  8E  F2  3F  STX  $3FF2
3FA9  BD  E5  3F  LDA  $3FE5,X
3FAC  20  B7  3F  JSR  $3FB7
3FAF  AE  F2  3F  LDX  $3FF2
3FB2  E8          INY
3FB3  88          DEY
3FB4  D0  F0      BNE  $3FA6
3FB6  60          RTS
3FB7  8D  F3  3F  STA  $3FF3
3FBA  8C  F5  3F  STY  $3FF5
3FBD  8E  F4  3F  STX  $3FF4
3FC0  20  D2  FF  JSR  $FFD2
3FC3  AE  F4  3F  LFX  $3FF4
3FC6  AC  F5  3F  LDY  $3FF5
3FC9  AD  F3  3F  LDA  $3FF3
3FCC  60          RTS
3FCD  A9  0D      LDA  #$0D
3FCF  20  B7  3F  JSR  $3FB7
3FD2  60          RTS
3FD3  20  E4  FF  JSR  $FFE4
3FD6  C9  20      CMP  #$20
3FD8  F0  01      BEQ  $3FDB
3FDA  60          RTS
3FDB  20  E4  FF  JSR  $FFE4
3FDE  C9  43      CMP  #$43
3FE0  D0  F9      BNE  $3FDB
3FE2  60          RTS
3FE3  00          BRK
3FE4  00          BRK
3FE5  49  4E      EOR  #$4E
3FE7  54          ???
3FE8  46  4C      LSR  $4C
3FEA  4F          ???
3FEB  53          ???
3FEC  54          ???
3FED  52          ???
3FEE  00          BRK
3FEF  03          ???
3FF0  06  00      ASL  $00
3FF2  00          BRK
3FF3  00          BRK
3FF4  00          BRK
3FF5  00          BRK
```

GLOSSARIO

ASSEMBLATORE

Particolare programma che provvede a convertire un altro programma scritto in forma mnemonica (linguaggio assembly), incomprensibile al microprocessore, in linguaggio macchina interpretabile da quest'ultimo ma di difficile manipolazione da parte del programmatore.

BINARIO

Sistema di numerazione in base 2. Utilizzato da quasi tutti i computers. Ogni cifra può assumere unicamente due valori: zero e uno. Attraverso l'impiego di più cifre binarie, strutturate secondo un ordine di grandezza (come nel sistema decimale), è possibile rappresentare qualunque valore numerico minore di $2\uparrow n$ (con $n =$ numero delle cifre binarie utilizzate).

BIT

Unità elementare d'informazione. Corrisponde ad una cifra binaria, la quale può assumere unicamente i valori zero e uno. Più bits concatenati permettono di rappresentare valori numerici superiori (vedere BINARIO, BYTE).

BUFFER

Area di memoria predisposta per un immagazzinamento temporaneo di dati specifici. Generalmente impiegato in relazione a funzioni di I/O (input/output).

BYTE

Unità di memoria del computer. Una locazione di memoria è in grado di contenere un byte d'informazione. Ogni byte è formato da otto bits contigui, e può immagazzinare un valore numerico compreso fra 0 e 255. Tale valore può rappresentare un carattere, un numero, oppure parte di un'istruzione del microprocessore. Come nel caso dei bits, più bytes raggruppati permettono di formare valori numerici superiori (vedere BINARIO, LOCAZIONE DI MEMORIA).

CARATTERE

Generalmente ogni simbolo alfanumerico visualizzabile sullo schermo at-

traverso la pressione di un tasto. Esistono inoltre particolari altri caratteri, sempre disponibili da tastiera, che rappresentano un'eccezione alla definizione precedente (vedere CARATTERI GRAFICI).

CARATTERI GRAFICI

Speciali caratteri caratterizzati soltanto da una forma grafica, i quali non rappresentano alcun significato simbolico, come lettere o numeri.

CODICE ASSEMBLY

Vedere LINGUAGGIO ASSEMBLY.

CODICE MACCHINA

Vedere LINGUAGGIO MACCHINA

DECIMALE

Sistema di numerazione in base 10. Corrisponde al sistema generalmente impiegato nella vita di tutti i giorni.

DISASSEMBLATORE

Particolare programma che provvede a visualizzare il linguaggio macchina in forma mnemonica, al fine di facilitare la lettura al programmatore (vedere LINGUAGGIO ASSEMBLY, ASSEMBLATORE).

DUMP

Visualizzazione del contenuto della memoria in forma numerica o letterale (caratteri ASCII).

ESADECIMALE

Sistema di numerazione in base 16. Viene spesso utilizzato in linguaggio macchina in quanto costituisce una forma più compatta del sistema binario. Un byte viene infatti rappresentato da otto cifre binarie, e soltanto da due cifre esadecimali (quattro bits rappresentano valori decimali da 0 a 15 ed esattamente una cifra esadecimale). In questo sistema vengono impiegate come cifre le lettere dalla A alla F per identificare i valori da dieci a quindici. Il simbolo $ viene generalmente impiegato per indicare che il numero è esadecimale.

INDIRIZZO DI MEMORIA

Ogni locazione di memoria viene progressivamente numerata da 0 a 65535, in modo da poterla facilmente reperire fra tutte quelle presenti nel computer. Un numero utilizzato per questo scopo viene appunto denominato "indirizzo di memoria".

INTERRUPT

Segnale elettronico inviato al microprocessore da una periferica o da un chip interno al computer, il quale provvede a notificargli che qualcosa sta accadendo nel mondo esterno (vedere MICROPROCESSORE).

LINGUAGGIO ASSEMBLY

Forma mnemonica utilizzata per la rappresentazione dei programmi in linguaggio macchina. Molto più facilmente interpretabile e manipolabile da parte del programmatore, tale forma letterale non riveste alcun significato per il microprocessore prima di essere trattata attraverso un assemblatore. Ogni grosso programma in linguaggio macchina viene costruito in questa speciale notazione mnemonica (vedere LINGUAGGIO MACCHINA, ASSEMBLATORE).

LINGUAGGIO MACCHINA

Linguaggio numerico di programmazione direttamente interpretabile ed eseguibile dal microprocessore. Un programma in linguaggio macchina corrisponde ad una stringa numerica che può essere memorizzata dal programmatore in notazione esadecimale, oppure costruita in forma mnemonica attraverso un assemblatore (vedere ASSEMBLATORE, LINGUAGGIO ASSEMBLY, MICROPROCESSORE).

LOCAZIONE DI MEMORIA

Termine utilizzato per caratterizzare un byte regolarmente associato ad un indirizzo (vedere BYTE).

MEMORIA

Struttura interna del computer suddivisa in singole celle ed utilizzata per l'immagazzinamento di numeri, istruzioni in linguaggio macchina e caratteri. Ogni sua cella è in grado di contenere un solo byte alla volta. La sua ampiezza si misura in Kbytes (1 Kbyte = 1024 bytes). Si suddivide in memoria RAM (a lettura e scrittura) e memoria ROM (a sola lettura).

MICROPROCESSORE

Unità centrale di elaborazione e controllo del computer. Può essere paragonato al cervello umano. Esso gestisce tutti i movimenti di dati, le decisioni ed i calcoli che si svolgono all'interno del calcolatore.

MODO GRAFICO

Speciale opzione che permette di visualizzare qualsiasi figura sullo schermo generata nei limiti della risoluzione grafica del computer (numero dei punti di schermo).

MODO TESTO

Contrariamente alla precedente, questa opzione permette di visualizzare unicamente i caratteri disponibili da tastiera (vedere SET DI CARATTERI).

PAGINA ZERO

Termine utilizzato per indicare le prime 256 locazioni di memoria del computer (da $00 a $FF).

SET DI CARATTERI

Insieme di tutti i caratteri disponibili che possono essere stampati sullo schermo di testo (vedere CARATTERE, CARATTERE GRAFICO, MODO TESTO).

VETTORE

Nome dato a due bytes della memoria RAM che contengono l'indirizzo iniziale di una routine ROM. La loro funzione consiste nel permettere al programmatore di accedere direttamente alle routines che rappresentano attraverso un salto indiretto. Essendo normali locazioni di memoria RAM, tali vettori possono inoltre essere modificati a piacimento in modo da farli puntare a particolari routines direttamente costruite dal programmatore.

ISTRUZIONI PER IL CORRETTO CARICAMENTO DEI PROGRAMMI SU CASSETTA PER COMMODORE 16

Scrivere LOAD o LOAD "nome programma" seguito dal tasto RETURN. Fare attenzione a digitare il nome esatto.

Avviare il registratore con il tasto play ed attendere l'avvenuto caricamento del programma.
Se il programma non è provvisto di AUTOSTART (partenza automatica), digitare RUN seguito dal tasto RETURN.
Pur eseguendo le istruzioni sopraindicate è possibile incontrare talvolta qualche difficoltà nel caricamento del programma. La prima cosa da fare è assicurarsi che la testina del vostro registratore risulti ben pulita, allineata e smagnetizzata.
Non modificate per nessun motivo l'allineamento della testina del registratore poiché risulta pressoché impossibile, se non si dispone di apparecchiature professionali, procedere alla taratura dello stesso; in tal caso rivolgetevi presso un laboratorio specializzato.

PROGRAMMI CONTENUTI NELLA CASSETTA

Nella cassetta allegata sono presenti i programmi proposti e commentati nell'Appendice 10.

GARANZIA CASSETTE SOFTWARE

Le cassette software originali, non manomesse, che presentassero eventuali difetti vanno spedite per la sostituzione a:
EDIZIONI JCE S.A.S.
Via dei Lavoratori, 124
20092 Cinisello Balsamo (MI)

NOTE

NOTE

NOTE

Scritto appositamente per gli utenti del Commodore 16, questo libro aprirà a tutti le porte dell'affascinante mondo del linguaggio macchina. In esso troverete esaudientemente commentate ed ampiamente illustrate tutte le istruzioni relative alla programmazione del microprocessore 7501, l'ultimo arrivato della famiglia 6502, ovvero il cuore del vostro C16. Se siete frustrati dalle limitazioni del BASIC e desiderate apprendere un linguaggio estremamente rapido, potente, compatto, allora questo libro è per voi. Anche senza disporre di alcuna esperienza di programmazione, "L'ABC DEL LINGUAGGIO MACCHINA PER IL C16" vi guiderà passo dopo passo alla scoperta del vostro computer e della sua potenza fino ad oggi soltanto teorica. Una completa spiegazione dei comandi di TEDMON è inoltre contenuta in un apposito capitolo, ai fine di permettervi la scrittura dei vostri programmi in linguaggio macchina fin dal primo approccio con il computer.

ISBN 88-7708-007-8

Cod. 9116

